МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Національний аерокосмічний університет ім. М. Є. Жуковського
“Харківський авіаційний інститут”

Р. Е. Пащенко

ПРОЕКТУВАННЯ БАЗ ГЕОДАНИХ

Конспект лекцій

Харків “ХАІ” 2018

УДК 004.65
П12

Рецензенти: д-р фіз.-мат. наук, с.н.с. В. К. Іванов,
д-р техн. наук, доц. В. А. Таршин

Пащенко, Р. Е.

П12 Проектування баз геоданих [Текст] : консп. лекцій / Р. Е. Пащенко. – Харків : Нац. аерокосм. ун-т ім. М. Є. Жуковського «Харків. авіац. ін.-т», 2018. – 156 с.

ISBN 978-966-662-624-3

Розглянуто відомості щодо методів проектування і роботи з базами даних у геоінформаційних системах (ГІС) і основи побудови реляційних баз даних і систем управління базами даних (СУБД). Наведено етапи розвитку проектування баз геоданих, способи подання даних у ГІС і структуру бази геоданих. Викладено основні поняття і принципи побудови реляційних баз даних, описано типову структуру сучасної СУБД, головні функції СУБД і внутрішню структуру реляційних СУБД, основи мови SQL і формування SQL-запитів, а також етапи і кроки проектування бази геоданих.

Для студентів, що навчаються за спеціальностями 193 «Геодезія та землеустрій» спеціалізації «Геоінформаційні системи і технології» та 103 «Науки про Землю» спеціалізації «Космічний моніторинг Землі».

Іл. 49. Табл. 9. Бібліогр. : 14 назв

УДК 004.65

ISBN 978-966-662-624-3

Зміст

Передмова

Конспект лекцій містить відомості про етапи розвитку проектування баз геоданих (БГД), подання даних у БГД, таких, як векторне для відображення просторових об'єктів, растрове – для безперервних півтонових зображень, сіткових тематичних даних і поверхонь, а також нерегулярні тріангуляційні мережі (TIN) – для трьохвимірних поверхонь; адреси і локатори для знаходження географічного положення об’єктів. Крім цього, розглянуто структуру бази геоданих, даних покриття і шейп-файлів, а також подання карт і шарів.

Особливу увагу приділено розкриттю основ побудови реляційних баз даних. Розглянуто ранні підходи до створення структури баз даних і базові поняття реляційних баз даних, а також наведено структуру реляційної бази даних. Наведено відомості щодо типової структури сучасної системи управління бази даних (СУБД) та її основні функції, такі, як безпосереднє управління даними у зовнішній пам'яті, управління буферами оперативної пам'яті, управління транзакціями, журналізація, підтримання мов баз даних. При цьому розглянуто структуру зовнішньої пам’яті та методи організації індексів, а також механізм управління транзакціями, методи серіалізації транзакцій та відновлення бази даних після збоїв. Коротко розглянуто основні операції реляційної алгебри і основи мови SQL під час формування SQL-запитів.

У конспекті наведено загальні відомості про принципи побудови та етапи створення моделей даних, розглянуто особливості проектування бази геоданих, а також відомості щодо етапів побудови бази геоданих, таких, як моделювання уявлень користувача, визначення об'єктів і відношень, вибір подання об'єктів, узгодження з моделлю даних бази геоданих, створення БГД у вигляді наборів географічних даних.

Конспект складається з вісімнадцяти лекцій, які охоплюють основний теоретичний матеріал навчальної дисципліни “Проектування баз геоданих”. Під час підготовки матеріалів лекцій автор користувався джерелами інформації, які наведені в кінці конспекту. Необхідно зазначити, що під час написання лекцій № 1 – 5 за основу було вибрано роботи [1, 3, 4, 6, 10], лекцій № 6 – 12 – роботи [5, 7, 8, 12, 13], лекцій № 13 – 15 – роботи [2, 14], а лекцій № 16 – 18 – роботи [1, 9].

Лекція № 1

ОСНОВНІ ПОНЯТТЯ БАЗ ГЕОДАНИХ

Навчальні цілі:

- розглянути основні поняття та визначення баз геоданих;
- вивчити структуру моделі даних покриття та подання даних у базах геоданих.

Навчальні питання:

1. Етапи розвитку проектування баз геоданих.
2. Подання даних у базах геоданих.

1.1 Етапи розвитку проектування баз геоданих

Для опису розміщення у просторі об'єктів і процесів використовують формальні моделі. На основі таких моделей будують всі географічні інформаційні системи (ГІС).

Формальна модель – це узагальнена та чітка система понять для опису і пояснення об'єктів і процесів.

Географічна модель даних визначає систему понять для опису і пояснення об'єктів і процесів на поверхні Землі.

Однією з найбільш поширених географічних моделей даних є карта. **Карта** являє собою масштабну модель реальності, яку створюють з використанням певних норм і правил (наприклад, картографічних проекцій, умовних знаків, надписів). За допомогою карти можна вирішити низку задач, наприклад, визначити відстань від Харкова до Києва або з'ясувати, які міста розташовані на берегах річки Дніпро. При цьому картографічна модель є засобом візуальної передачі географічних відомостей, наприклад, про характер рельєфу (наскільки він пересічений) або про місцезнаходження півночі. Через те, що під час створення карт використовують загальновідомі правила (наприклад, блакитні лінії – це річки, північ знаходиться вгорі і т.д.), карти є ефективними і широко застосовуваними. Коли фахівці починають створювати або модифікувати власні географічні моделі даних, вони можуть визначати свій набір понять і відношень, але такі нововведення мають бути зрозумілими всім користувачам. Крім того, ці поняття мають забезпечувати можливість відтворювати географічну інформацію засобами комп'ютерної системи.

Таким чином, модель географічних даних є деякою абстракцією реального миру, основаною на використанні набору об'єктів даних, що забезпечують відображення, запити, редагування і аналіз карт.

Під час створення ГІС використовують знання з різних наукових дисциплін: математики (геометрії і теорії мереж), фізики (теорії дискретизації і вимірювань), дистанційного зондування, інформаційних

технологій (моделювання і управління багатокористувальними базами даних).

Розвиток ГІС і проектування баз геоданих (БГД) пройшли декілька етапів, коротко розглянемо їх.

Перший етап розпочато у 1970-х роках і пов'язано з тим, що картографічні моделі створювалися за допомогою систем автоматизованого проектування (САПР) загального призначення, тобто для створення електронних карт не використовувалися спеціалізовані програмні засоби. У моделі даних САПР для зберігання географічних даних використовували двійкові файли, в яких зберігали точки, лінії і полігони. У цих файлах також містилася і деяка атрибутивна інформація, в основному це були надписи.

У перших комп'ютеризованих системах, що застосовувалися для створення картографічної продукції, використовувалися векторні пристрої графічного відображення (монітори). За допомогою цих засобів відображали тільки прямі лінії. Крім того, растрові карти друкувалися на алфавітно-цифрових чорно-білих друкуючих пристроях. При цьому потрібна ступінь зачорніння створених об’єктів досягалася застосуванням різних символів, а також друком кожного рядка у декілька проходів. У зв’язку з цим основні напрямки розвитку електронних картографічних систем були пов’язані з удосконаленням графічного апаратного забезпечення та картографічних програм, за допомогою яких можна було створювати карти прийнятного для картографії вигляду.

Другий етап розвитку ГІС та проектування БГД пов'язано зі створенням у 1981 р. в Інституті досліджень природних систем (Environmental Systems Research Institute, Inc., ESRI) першої комерційної ГІС ArcInfo. У програмному пакеті ArcInfo була вперше реалізована модель даних покриття, яку також називають георeляційною моделлю даних.

Модель даних покриття відрізняється від попередніх наявністю двох ключових особливостей. По-перше, в цій моделі просторова інформація скомбінована з атрибутами. Просторові дані зберігаються в індексованих двійкових файлах, оптимізованих для відображення і доступу. Атрибутивні дані зберігаються в таблицях, де число рядків дорівнює числу векторних об'єктів у двійкових файлах, і вони з'єднуються за допомогою загального поля ідентифікатора. По-друге, в моделі є можливість зберігати топологічні зв'язки між векторними об'єктами. Це означає, що запис просторових даних у кожній лінії містить інформацію про те, які вузли її утворюють і які лінії з'єднані. Крім того, запис також містить інформацію про те, які полігони знаходяться справа і зліва від лінії.

З введенням моделі даних покриття у користувачів з’явилася можливість модифікувати таблиці векторних об'єктів. При цьому можна не тільки додавати поля, але й пов'язувати таблиці атрибутів з таблицями зовнішніх баз даних.

Однак модель даних покриття мала деякі обмеження. Наприклад, векторні об'єкти в моделі об’єднуються в однорідні набори точок, ліній і полігонів із загальною поведінкою. При цьому поведінка лінії, яка позначає дорогу, є такою ж, як і поведінка лінії, що позначає річку. При цьому загальна поведінка об’єктів, яка підтримувалася моделлю даних покриття, фіксувала тільки топологічну цілісність набору даних. Наприклад, якщо додавалася лінія, що перетинала полігон, то він автоматично поділявся на два полігони. У той же час бажано підтримувати також і спеціальну поведінку об'єктів реального миру (річок, доріг та ін.). Наприклад, річки течуть вниз по схилу, а при злитті водотоків витрата об'єднаного водотоку дорівнює сумі витрат водотоків, що його створюють. Ще один приклад – коли перетинаються дві дороги, в цьому місці обов'язково мають бути перехрестя, тунель або естакада.

Третій етап настав у 2000-х роках з початком нового тисячоліття. Його початок можна віднести до виходу чергової версії програмної продукції компанії ESRI – ГІС ArcInfo 8. Нова підсистема ARCGIS у її складі мала чисто графічний інтерфейс і працювала з новим сховищем даних – базою геоданих. Поряд з новим графічним інтерфейсом у ArcInfo 8, як і раніше, підтримувалися дані покриття ARC/INFO і шейп-файли. На початку застосування ГІС ArcInfo 8 база геоданих розглядалася користувачами як перенесення зберігання даних з окремих файлів у базу даних. Однак у подальшому починають активно розвивати модель даних нового сховища – бази геоданих. З розвитком цієї моделі сімейство продуктів ESRI стали називати ARCGIS, а ArcInfo увійшло як його частина. З появою ARCGIS 9 з’явилася нова можливість системи – моделювання у базі геоданих.

Таким чином, відбулася зміна геоінформаційної парадигми: перейшли від моделювання карт до моделювання реальності. Модель реальності вже не буде картою. В ній можуть знаходитися не тільки просторові об'єкти, але й непросторові. Наприклад, у моделі даних кадастру земельна ділянка – це просторовий об'єкт, а його власник – непросторовий. Тому у базі геоданих табличні (об'єктні) класи підтримуються так само, як і класи просторових об'єктів.

База геоданих – це сховище географічних даних, об’єднаних у набори географічних даних, класи просторових об'єктів, об'єктні класи і класи відношень.

На сучасному етапі найбільш поширеними зарубіжними системами, в яких застосовуються БГД, є програмні засоби ArcInfo, ArcView, MapInfo Professioal, ГеоГраф, Панорама. Коротко розглянемо призначення цих програмних продуктів.

ArcInfo (розробник – фірма ESRI, США) є найбільш потужною повнофункціональною ГІС. ArcInfo складається з двох програмних пакетів: ArcInfo Workstation та ArcInfo Desktop. Ці програмні пакети можуть встановлюватися окремо один від одного.

Програмний пакет ArcInfo Workstation містить три базових модулі:

- ArcMap – для відображення, редагування і аналізу даних;
- ArcCatalog – для забезпечення доступу до даних і управління ними;
- ArcToolbox – для розширеного просторового аналізу, управління проекціями і конвертації даних.

Призначення додаткових модулів є такими:

- Arc COGO – для роботи з геодезичними даними;
- Arc GRID – для аналізу і управління безперервно розподіленими числовими і якісними ознаками, що наводяться у вигляді регулярних моделей, а також моделювання складних процесів;
- ARC TIN – для моделювання топографічних поверхонь;
- Arc NETWORK – для моделювання і аналізу топологічно поєднаних об'єктів у вигляді просторових мереж, оцінювання ресурсів та управління ними, розподіленими за мережами і процесами у таких мережах.

ArcInfo забезпечує створення геоінформаційних систем, створення і ведення земляних, лісових, геологічних та інших кадастрів, проектування транспортних мереж, оцінювання природних ресурсів.

ArcView (розробник – фірма ESRI, США) являє собою настільну ГІС, яка дозволяє користувачеві вибирати й переглядати різноманітні геодані, редагувати їх, а також аналізувати й подавати у різних видах.

ArcView підтримує реляційні СУБД, має розвинену ділову графіку (форму перегляду, табличну форму, форму діаграм, створення макету), передбачає створення професійно оформленої картографічної інформації та розроблення власних додатків.

MapInfo Professional (розробник – фірма MapInfo Corp., США) спеціально спроектована для оброблення і аналізу інформації, що має адресну або просторову прив'язку.

У програмі MapInfo можна здійснювати:

- пошук географічних об'єктів;
- роботу з базами даних;
- розрахунок геометричних параметрів (площ, довжини, периметрів, об'ємів);
- побудову буферних зон навколо будь-якого об'єкта або групи об'єктів;
- застосування розширеної мови запитів SQL (запити ґрунтуються на виразах, виконують об'єднання, відображають доступні поля, дозволяють робити підзапити, об'єднання з декількох таблиць та географічні об'єднання);
- комп'ютерний дизайн і підготовку до видання картографічних документів.

ГеоГраф (розробник – Центр інформаційних досліджень Інституту географії РАН, Росія) дозволяє створювати електронні тематичні атласи та композиції карт на основі шарів цифрових карт і пов'язаних з ними таблиць атрибутивних даних.

ГеоГраф дозволяє формувати просторові об'єкти у вигляді косметичних шарів з прив'язкою до них таблиць атрибутивних даних, забезпечує управління атрибутивними даними, у тому числі приєднанням таблиць, а також редагування, вибір, сортування, запити за зразком та ін. Крім того, ГеоГраф має засоби електронного тематичного картографування та ін.

ГІС "Панорама" (розробник – ЗАТ КБ "Панорама", Росія) призначена для побудови та редагування цифрових карт, оброблення даних дистанційного зондування Землі, дозволяє виконувати різні вимірювання та проводити розрахунки, здійснювати оверлійні операції та будувати 3D-моделі. ГІС "Панорама" має засоби підготовки графічних документів в електронному й друкарському вигляді, а також інструментальні засоби для роботи з базами даних. Існують професійна й настільна версії ГІС "Панорама".

Існують також професійні багатофункціональні інструментальні ГІС, що забезпечують можливість безпосереднього оброблення даних дистанційного зондування. Найбільш поширеними є ERDAS IMAGINE і ERMapper.

ERDAS IMAGINE (розробник – Leica) – програмний пакет, який спеціально створено для оброблення і аналізу даних дистанційного зондування. ERDAS IMAGINE містить повний набір інструментів для аналізу даних, отриманих з будь-якого джерела, і подання результатів у різних формах (друкарських картах, тривимірних моделях та ін.).

ERDAS IMAGINE побудовано за модульним принципом. Основними його компонентами є IMAGINE Essential, IMAGINE Advantage та IMAGINE Professional. Цей пакет надає широкі можливості з візуалізації та імпорту даних (більше 100 форматів), геометричної корекції, а також забезпечує проведення перетворень, що покращують зображення, ГІС-аналіз і дешифрування знімків. Крім того, ERDAS IMAGINE має широкий набір інструментів оброблення зображень і побудови алгоритмів просторових обчислень і створення карт.

ER Mapper (розробник – ER Mapper) забезпечує оброблення великих обсягів фотограмметричної інформації, здійснює тематичне картографування (природних ресурсів, лісового господарства та ін.), а також має засоби, що забезпечують друкування карт, візуалізацію тривимірного зображення та ведення бібліотеки алгоритмів.

1.2 Подання даних у базах геоданих

Кожен проект бази даних ГІС ґрунтується на прийнятті рішення про те, які географічні подання будуть використані для кожного набору даних.

База геоданих може містити географічні дані у чотирьох формах:

1. Векторні дані для відображення просторових об'єктів.
2. Растрові дані для двовимірного відображення безперервних

півтонових зображень, сіткових тематичних даних і поверхонь.

3. Нерегулярні тріангуляційні мережі (TIN) для тривимірного відображення поверхонь.

4. Адреси і локатори для знаходження географічного положення об’єктів.

Багато реальних географічних об'єктів мають форму з чітким контуром. Для їх відображення, як правило, використовують **векторне подання.** Векторні дані точно і компактно описують форму просторових об'єктів у вигляді впорядкованого набору координат з асоційованими атрибутами. Таке подання підтримує деякі геометричні операції, наприклад, розрахунок довжини або площі, знаходження суміжних просторових об'єктів або таких, що знаходяться поблизу.

Векторні об'єкти частіше за все класифікують за їх розмірністю: точки, лінії, полігони.

Точками відображають географічні об'єкти дуже малих розмірів, зображати їх лініями або полігонами неможливо. Точки є фігурами без вимірювань, вони зберігаються як одна пара координат (*x*, *y*) і набір атрибутів.

Лініями відображають дуже вузькі географічні об'єкти, зображати їх полігонами неможливо. Лінії є фігурами з одним вимірюванням, вони зберігаються як упорядкований набір пар координат (*x*, *y*) і набір атрибутів. Сегменти ліній можуть бути прямими, округлими, еліптичними або сплайнами.

Полігонами зображають широкі (великі) географічні об'єкти. Полігони є фігурами з двома вимірюваннями. Кожен полігон зберігається як набір сегментів ліній, що є його межею.

Існує ще один тип векторних даних – це **підпис (анотація)**. Вони є описами, які пов'язані з просторовими об'єктами і відображають назви або атрибути.

Векторні дані у базі геоданих мають структуру, яка управляє зберіганням просторових об'єктів відповідно до їх розмірності і відношень. Набором класів об'єктів є контейнер просторових елементів (просторових об'єктів), непросторових елементів (об'єктів) і відношень між ними. Топологічні відношення описуються за допомогою геометричних мереж і плоских топологій. База геоданих містить також правила перевірки коректності й атрибутивні домени, за допомогою яких гарантують, що при створенні й оновленні просторових об'єктів їх атрибути залишаться правильними відносно поєднаних просторових і непросторових об'єктів.

Велика кількість даних може надходити у БГД у сітковій формі, що пов’язано з тим, що знімальні камери та системи дистанційного зондування Землі записують дані у вигляді піксельних значень на двовимірній рівномірній сітці, яка зветься **растром**. Для відображення таких даних, як правило, використовують **сіткове подання за допомогою растрів.**

Основним елементом растру є **чарунка** (піксел, англ. pixel, picture element). Числове значення чарунки може відображати різноманітні дані: величину відбитого світлового потоку у деякій області спектра, колір на фотографії, тематичний клас (наприклад, тип рослинності), висоту поверхні та ін.

Для 3D-відображення поверхонь використовують **подання за допомогою нерегулярної тріангуляційної мережі (TIN).** База геоданих зберігає TIN у вигляді інтегрованого набору вузлів із значеннями висоти і трикутників із сторонами, що з'єднують вузли. Висоту (*z*-значення) можна інтерполювати в будь-якій точці у межах географічного екстента (області поширення) TIN. За допомогою моделі даних TIN можна виконувати різні види аналізу поверхні, наприклад, дослідження водозборів, оцінювання видимості довільних точок поверхні із заданої точки спостереження, показ особливостей поверхні, таких, як хребти, схили або вершини. За допомогою моделі TIN також можна зображати рельєф місцевості.

Набір даних TIN отримують у результаті здійснення тріангуляції (об'єднання трикутниками) набору нерівномірно розташованих точок із двома значеннями, які описують характер поверхні. TIN найчастіше використовують для моделювання земної поверхні, але її також можна застосовувати для відображення безперервного розподілу будь-якого природного фактора, наприклад, концентрації хімічної речовини.

Адреси й локатори частіше за все використовують для вирішення географічних задач пошуку адрес місць знаходження об'єктів. Локатори містять інформацію, яка дозволяє створювати просторові об'єкти за місцеположеннями, що визначаються цими локаторами.

Усередині бази геоданих в усіх класах просторових об'єктів використовують загальну систему координат – це є необхідною умовою задання топологічних зв'язків.

Корпоративні бази даних містять багато записів з адресами і місцеположеннями, що задаються іншими способами. Локатори містять інформацію, яка дозволяє створювати просторові об'єкти у цих місцеположеннях з метою відображення на карті.

Запитання для самоперевірки

1. Чим відрізняються формальна й географічна моделі ?
2. Чим характеризуються різні етапи розвитку проектування БГД ?
3. Які закордонні програмні засоби роботи з базами геоданих є найбільш поширеними ?
4. Які географічні подання використовують у базах геоданих ?
5. Як класифікують векторні об'єкти у базах геоданих ?
6. Що є основним елементом растра і яка інформація в ньому може зберігатися ?
7. Що являє собою подання за допомогою нерегулярної тріангуляційної мережі (TIN) ?

Лекція № 2

СПОСОБИ ПОДАННЯ ДАНИХ У ГІС

Навчальні цілі:

- розглянути технічне (апаратне), програмне й інформаційне забезпечення ГІС;
- вивчити способи моделювання поверхонь і підходи до вибору способу подання просторових даних.

Навчальні питання:

1. Технічне (апаратне), програмне й інформаційне забезпечення ГІС.
2. Моделювання поверхонь.
3. Вибір способу подання просторових даних.

2.1 Технічне (апаратне), програмне й інформаційне забезпечення ГІС

Кожний користувач ГІС виконує конкретні завдання, пов’язані, наприклад, з дослідженням або обліком природних ресурсів. Поставлені завдання, як правило, стосуються певних територій, що вирізняються природними умовами і ступенем їх освоєння. Це обумовлено наявністю певних вимог до технічного, програмного та інформаційного забезпечення ГІС.

Технічне забезпечення – це комплекс апаратних засобів, які використовуються під час функціонування ГІС.

До апаратних засобів належать:

- робоча станція або персональний комп'ютер (ПК);
- пристрої введення і виведення інформації;
- пристрої оброблення і зберігання даних;
- засоби телекомунікації.

Робоча станція або персональний комп'ютер є основою будь-якої геоінформаційної системи. Їх призначено для управління роботою ГІС, а також для оброблення даних, одержаних в результаті проведення обчислювальних або логічних операцій. Сучасні ГІС можуть оперативно обробляти величезні масиви інформації і візуально подавати їх результати.

З покращенням пропускної спроможності мереж бажаною конфігурацією ГІС у масштабі підприємства стала багаторівнева архітектура «клієнт-сервер», тобто локальна мережа підприємства. Об’єднання комп’ютерів і локальних мереж у глобальну мережу Інтернет стало важливим інструментом доступу до даних взагалі та геоданих зокрема.

До **пристроїв уведення даних** належать: клавіатура ПК; різноманітні сканери; зовнішні комп'ютерні системи. Крім того, просторові дані можуть бути отримані електронними геодезичними приладами, а також у результаті оброблення знімків на аналогових або цифрових фотограмметричних приладах і станціях. Для визначення місцеположень людей та об’єктів у режимі реального часу застосовують системи глобального позиціювання (GPS, ГЛОНАС та ін.).

Пристрої виведення даних забезпечують наочне подання результатів. Результати оброблення даних можуть бути виведені на екран монітора ПК, а також можуть бути роздруковані на принтері або на іншому графічному пристрої (наприклад, плоттері) у вигляді карт або інших графічних зображень. Крім того, пристрої виведення інформації мають забезпечувати експорт даних у зовнішні системи.

Пристрої для оброблення і зберігання даних є елементами системного блока ПК, який містить центральний процесор, оперативну і постійну пам'ять, зовнішні пристрої пам'яті, а також придатний інтерфейс користувача.

Програмне забезпечення – сукупність програмних засобів, які реалізовують функціональні можливості ГІС, а також документи, що необхідні при їх експлуатації. Програмне забезпечення ГІС містить **базові й прикладні програмні засоби.**

До **базових програмних засобів** належать: операційні системи (ОС), програмні середовища, мережне програмне забезпечення та системи управління базами даних (СУБД). Операційні системи призначені для управління ресурсами ПК і процесами, що використовують ці ресурси. У наш час найбільше розповсюдження отримали ОС Windows та Unix. Для роботи з просторовими і атрибутивними даними у будь-якій ГІС необхідно застосовувати програмне забезпечення для управління базами даних, а також модулі управління засобами введення і виведення даних, системи візуалізації даних і модулі для виконання просторового аналізу.

Прикладні програмні засоби призначені для виконання спеціалізованих завдань у вигляді окремих спеціалізованих програм або утиліт.

Основною ідеєю програмного забезпечення ГІС є те, що воно фактично є системою управління базою географічних даних. Бази геоданих реалізуються на основі комерційних систем управління реляційними або об'єктно-реляційними базами даних. При цьому відбувається нарощування можливостей комерційних СУБД, які підтримують: резервне копіювання; визначення схеми бази даних (БД); управління транзакціями та використання інструментів адміністрування системи. ГІС розширює реляційну БД ефективним зберіганням географічних даних, виробництвом карт і виконанням просторового аналізу.

Програмне забезпечення ГІС додає деякі функції до СУБД, основними з яких є такі:

- можливість зберігання геометричної форми просторових об'єктів прямо у стовпці БД;
- застосування засобів опису шарів карт і вказівок щодо методів відображення;
- візуалізація шарів на основі значень атрибутів;
- використання інфраструктури, що підтримує створення простих і складних карт і значно спрощує звичайні завдання щодо складання карт;
- створення і зберігання топологічних відношень між просторовими об'єктами, включаючи мережні структури та інтегровану полігональну топологію;
- двовимірне просторове індексування для забезпечення швидкого вибору географічних об'єктів;
- визначення просторових відношень, таких, як близькість, примикання, перекриття й просторове порівняння за допомогою набору операторів;
- застосування великої кількості інструментів для виконання просторових запитів, наприклад, трасування мереж та оверлійного аналізу;
- забезпечення управління потоком робіт, що дозволяє багатьом користувачам одночасно редагувати географічні дані та управляти версіями даних.

Можна вважати, що ГІС являє собою просторово розширену СУБД.

Інформаційне забезпечення – сукупність масивів інформації, систем кодування та класифікації інформації.

До інформаційного забезпечення належать реалізовані рішення щодо видів, обсягів, розміщення та форм організації інформації. Крім того, воно містить пошук та оцінювання джерел даних, набір методів введення даних, проектування БД, їх ведення та метасупроводження.

Просторові дані в сучасних ГІС зберігаються за допомогою їх поділення на окремі шари. Така багатошарова структура електронної карти дозволяє об'єднати і відобразити набагато більшу кількість інформації порівняно із звичайними картами. При цьому існує гнучкий механізм управління шарами.

2.2 Моделювання поверхонь

У сучасних ГІС поверхні можна моделювати трьома способами: ізолінією (ізогіпсою); растром; нерегулярною тріангуляційною мережею.

Моделювання за допомогою векторних даних. Векторні дані відображають просторові об'єкти у вигляді точок, ліній, полігонів і краще всього підходять для дискретних об'єктів з певними формами і межами.

Просторові об'єкти мають точну форму і положення, атрибути і метадані, а також використовувану поведінку (метод).

Моделювання поверхонь за допомогою векторних даних здійснюється з використанням ізоліній (горизонталей або ізогіпсів).

Ізолінія являє собою лінію, яка з'єднує точки з одним і тим же значенням ознаки, що зображується. У разі відображення рельєфу земної поверхні ізолінії називаються **горизонталями** або **ізогіпсами**, вони з'єднують точки однакової висоти. Ізогіпси є найпоширенішим джерелом інформації про рельєф для більшості користувачів карт (рисунок 2.1).

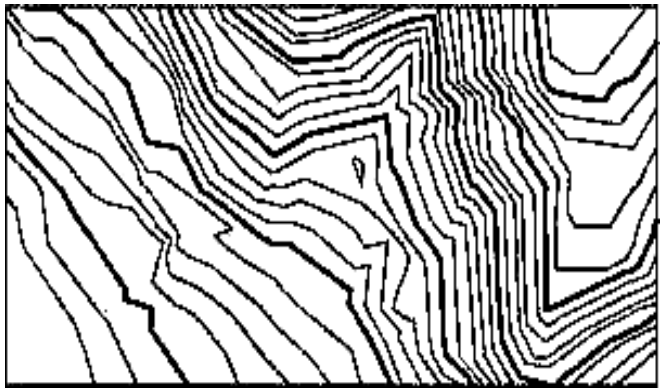


Рисунок 2.1 – Зображення поверхні
за допомогою ізогіпсів

Однією з переваг зображення поверхонь за допомогою ізогіпсів є зручність для сприйняття інформації. Щільність розташування ізогіпсів дає ясне зорове уявлення про ділянки земної поверхні з великою крутизною (схилом). Різка зміна напряму ізогіпсів є ознакою тальвегу або гребеня.

У той же час необхідно відзначити, що зображення поверхонь за допомогою ізогіпсів є незручним під час комп'ютерного моделювання поверхонь. Велика кількість точок на ізогіпсах не утворює задовільного набору даних для комп'ютерного відображення поверхні. Крім того, якщо є необхідність перетворення ізогіпсів у растри або TIN доводиться виконувати трудомістку роботу з видалення артефактів, що виникають при такому перетворенні.

Моделювання за допомогою растрів. Растрами відображають безперервні дані або зображення. Кожна чарунка (або піксел) у растрі характеризує вимірювану величину. Найбільш типовим джерелом для набору растрових даних є космічне зображення або аерофотознімок. Набір растрових даних також може бути фотографією об'єкта, наприклад, будівлі. Растри найкраще використовувати для зберігання і роботи з безперервними даними, такими, як висота, рівень ґрунтових вод, концентрація забруднюючих речовин і рівень шумового фону.

Моделювання поверхонь за допомогою растрів здійснюється при використанні даних про рельєф, які надаються у формі регулярної сітки зі значеннями висот у чарунках (рисунок 2.2). Прикладом цього можуть бути

дані Геологічної служби США, що надаються у вигляді продукту під назвою Digital Elevation Model (DEM) – цифрова модель рельєфу.

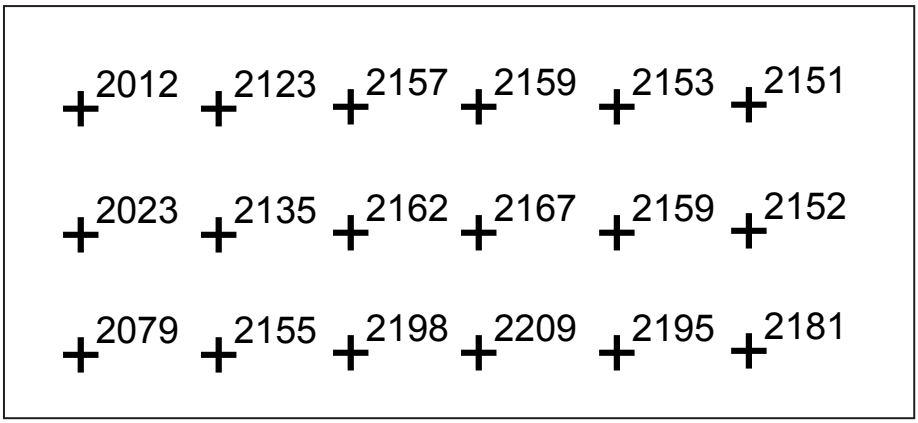


Рисунок 2.2 – Зображення поверхні за допомогою растру

Набір растрових даних може зберігати рівномірно розподілені відмітки висот. Кожна чарунка растра зберігає своє значення висоти (див. рисунок 2.2). З використанням набору растрових даних зі значеннями висоти можна розрахувати висоту будь-якої точки на поверхні та отримати набір ізоліній (ізогіпсів).

Перевагами зображення поверхонь за допомогою растрів є, по-перше, простота побудови моделі, по-друге, компактне зберігання даних, по-третє, велика кількість вже добре розроблених алгоритмів для оброблення растрових даних. Крім того, зберігається багато моделей рельєфу у растровому форматі і вони є недорогими.

Поряд з перевагами існує низка недоліків растрового подання. По-перше, жорстка структура сітки не адаптується до змін рельєфу. По-друге, при інтерполяції наявні початкові дані не завжди збігаються з сіткою з регулярним кроком, що призводить до помилок відображення. По-третє, растрове зображення лінійних об'єктів не забезпечує необхідної точності при багатьох застосуваннях.

Моделювання за допомогою нерегулярної тріангуляційної мережі (TIN – triangulated irregular network). Нерегулярна тріангуляційна мережа є корисним та ефективним способом тривимірного відображення поверхні ділянки землі, а також підтримує перспективні зображення. Крім того, можна накласти фотографічне зображення поверх TIN для фотореалістичного відображення рельєфу місцевості. Найбільш часто й з високою ефективністю TIN використовують для моделювання водозборів, лінії прямої видимості, крутизни, експозиції, хребтів і річок та вимірювання об'ємів. Моделювання поверхонь за допомогою TIN є найбільш ефективним і точним, якщо зображуються безперервні поверхні.

Як правило, набір даних TIN формується поетапно. Спочатку за допомогою фотограмметричних інструментів, приймачів GPS або іншим способом отримують набір точок з трьома координатами *x*, *y*, *z*. Далі

визначають лінії перегину там, де форма поверхні різко міняється , що відповідає на поверхні землі гребеням, тальвегам та ін. Після цього визначають області виключення горизонтальних ділянок, як правило, це поверхня води. На кінцевому етапі за цими точковими даними за допомогою програмного забезпечення ГІС створюють мережу трикутників, яка називається тріангуляцією Делоне (рисунок 2.3). При цьому грані у TIN створюються як можна більш схожими на рівносторонні трикутники.

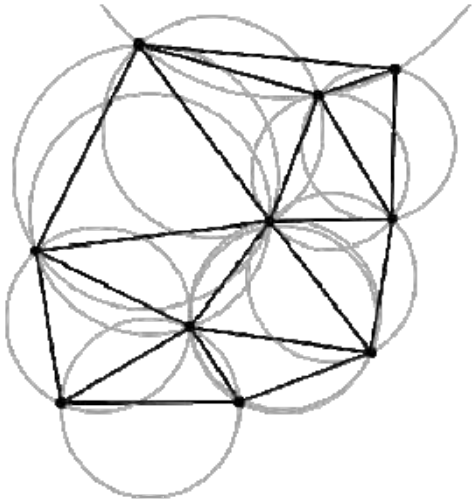


Рисунок 2.3 – Тріангуляція Делоне

Нагадаємо, що тріангуляція Делоне – це така тріангуляція, при якій жодна з точок набору **S** не потрапляє всередину жодного з описаних навколо отриманих трикутників кіл (див. рисунок 2.3). Розроблено велику кількість алгоритмів побудови тріангуляції Делоне, наприклад, алгоритми прямої побудови, алгоритми злиття, двопрохідні алгоритми, ітеративні алгоритми та ін. Кожен з алгоритмів вирізняється як методом побудови, так і трудомісткістю реалізації.

Тріангуляція Делоне не є оптимальною, але вона дозволяє побудувати набір трикутників, які наближаються до рівнокутних. Така тріангуляція має важливі властивості, а саме:

- максимальну суму мінімальних кутів всіх своїх трикутників серед всіх можливих тріангуляцій при заданому наборі точок;
- мінімальну суму радіусів кіл, описаних біля трикутників, серед всіх можливих тріангуляцій при заданому наборі точок.

Грань TIN являє собою трикутник, який розміщено у трьохвимірному просторі (рисунок 2.4). Грань визначає площину, її нахил і напрямок. Для будь-якої точки з координатами (*x*, *y*) за допомогою TIN можна розраховувати висоту, спочатку знаходячи ту грань, що містить точку, і потім інтерполюючи значення висоти в її межах.

Поверхню на базі нерегулярної тріангуляційної мережі будують за допомогою елементів TIN. Розбиття на трикутники проводять за багатьма масовими точками, кожна з яких утворює кортеж *x*, *y*, *z*. **Масова точка** – це точка з відомими координатами *x*, *y*, *z*. Як було зазначено вище, їх можна

отримати за допомогою фотограмметричних інструментів, приймачів GPS або з конвертованих даних.

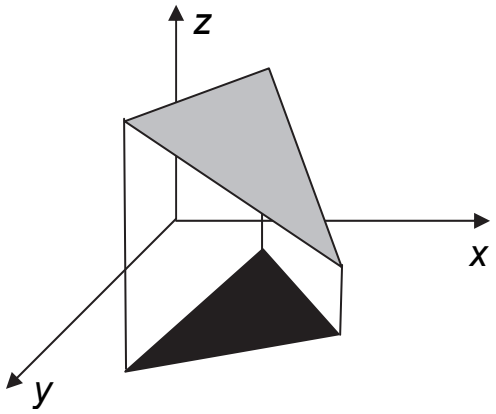


Рисунок 2.4 – Зображення грані TIN

Лінії перегину обкреслюють різкі неоднорідності рельєфу місцевості. Лінії перегину використовують, наприклад, для моделювання водотоків, гребенів, країв будівельних майданчиків, а також для позначення тальвег та інших лінійних неоднорідностей.

Області виключення позначають полігони з однаковою висотою, такі, як озера або рівнини, тобто строго горизонтальні ділянки, найчастіше це водні поверхні.

Межа проекту дозволяє виключати поверхню за межами області інтересу. Це може бути важливим при розрахунку об'ємів.

Зазначені елементи нерегулярної тріангуляційної мережі (TIN) показано на рисунку 2.5.

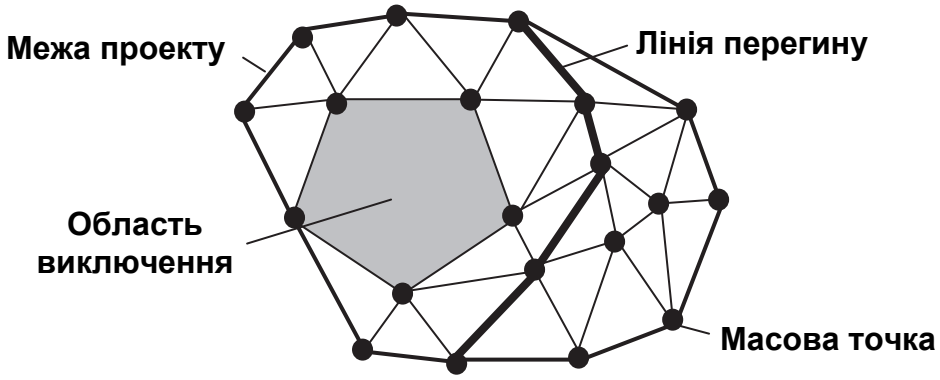


Рисунок 2.5 – Елементи нерегулярної тріангуляційної мережі (TIN)

Програмне забезпечення щодо роботи з TIN містить багато функцій аналізу поверхонь. За допомогою TIN можна виконати такі аналітичні завдання щодо аналізу поверхні, яка моделюється:

- обчислити висоту, схил та експозицію (напрям схилу за компасом) будь-якої точки поверхні;

- за мережею тріангуляції побудувати ізолінії з використанням лінійної або гладкої інтерполяції поліномом п'ятого ступеня;

- визначити діапазон значень висоти поверхні;

- обчислити статистичні дані поверхні землі, наприклад, її об'єм щодо опорної площини, середній схил, площу та периметр;

- показати вертикальний профіль поверхні вздовж указаної ламаної лінії;

- виконати розрахунки об'єму ґрунту для проектів дорожніх робіт, наприклад, так, щоб об'єм, що виймається на одній ділянці, дорівнював об'єму, що відсипається на іншій ділянці;

- визначити ділянки поверхні, які видно із заданої точки.

Побудова нерегулярної тріангуляційної мережі є ефективним методом зображення поверхонь, оскільки щільність точок у будь-якій частині поверхні може бути пропорційною пересіченості рельєфу. Для плоскої рівнини достатньо малої щільності точок. Гористий рельєф потребує високої щільності точок, особливо на ділянках швидких перепадів висот.

2.3 Вибір способу подання просторових даних

На початку процесу проектування бази даних ГІС користувач має визначити, які географічні подання будуть використані для кожного набору даних. Під час прийняття цього рішення користувачеві слід проаналізувати ряд аспектів.

По-перше, визначити, чи є готові до використання дані, тобто на початку проектування БГД користувач досліджує всі наявні географічні дані. Якщо є готові і придатні до використання дані (векторні, растрові або дані TIN), то доцільно прийняти рішення про їх застосування. Далі користувач визначає, чи достатньо цих даних для виконання поставленого завдання. Якщо даних недостатньо, то приймається рішення про створення нових даних іншими шляхами, наприклад, за допомогою фотограмметричних інструментів, приймачів GPS або конвертуванням даних.

По-друге, користувач аналізує, що є головним під час моделювання – просторові об'єкти або їх місцеположення.

Векторне подання буде кращим у разі моделювання окремо розташованих об'єктів з чіткими межами, поведінкою і атрибутами.

Растрові й тріангуляційні дані доцільно використовувати, якщо моделюються безперервні об'єкти або явища, що змінюються разом з місцеположенням. При цьому за допомогою растрових даних моделюються області з використанням регулярної сітки (рівномірного кроку вибірки атрибутів), а тріангуляційні дані – із застосуванням точок і значень зі змінною щільністю.

По-третє, користувач має визначити, яку точність просторового розміщення об'єктів необхідно забезпечити.

При високих вимогах до точності розміщення просторових об'єктів доцільно вибрати векторне подання даних. При цьому спрощується вибір та ідентифікація просторових об'єктів у результаті збереження точних значень їх координат.

Растрові дані забезпечують точність визначення положення просторових об'єктів, яка обмежується відстанню між сусідніми чарунками (пікселами) растру, тобто не може бути більшою, ніж розміри однієї чарунки.

Під час застосування тріангуляційних даних з високою точністю визначаються тільки положення масових точок і ліній перегину. Необхідно також зазначити, що застосування растрових і тріангуляційних даних не забезпечує високої точності визначення місцеположення та форми просторових об'єктів.

Наступним кроком є визначення типів просторових об'єктів, які необхідно моделювати.

Просторові об'єкти, що створюються людиною (штучні просторові об'єкти), як правило, мають чіткі форми, зображуються прямими лініями і дугами кола. Крім того, положення таких об'єктів часто визначається з геодезичною точністю. У зв’язку з цим під час моделювання штучних просторових об'єктів краще використовувати векторні дані. Векторне подання також можна застосовувати для зображення деяких природних просторових об'єктів, наприклад, річкових систем, які являють собою лінії.

Застосування растрових даних є найкращим способом зображення великих просторових об'єктів. Растрове подання також доцільно використовувати, якщо розміри просторових об'єктів змінюються у часі або форма їх не визначена, наприклад, поширення забруднюючих речовин у повітрі або розвиток пожежі у часі.

Тріангуляційне подання є найбільш ефективним під час моделювання просторових об'єктів, що відображають особливості форми земної поверхні, наприклад, гірських вершин, ліній гребенів або водотоків.

У ході вибору географічного зображення просторових об’єктів важливим є з’ясування того, який тип аналізу буде потрібним користувачеві. Він може застосовувати векторні дані, якщо йому в подальшому буде необхідно визначати оптимальне розміщення підприємств з точним місцеположенням; досліджувати потоки у мережі; управляти земельними записами; пов'язувати поштові адреси з місцеположенням об’єктів на карті або виконувати запит просторових об'єктів на карті.

Растрові дані доцільно застосовувати, якщо у подальшому буде необхідно аналізувати поширення нечіткого просторового об'єкта, наприклад, забруднення у повітрі. Крім того, таке подання дозволяє

визначити близькість просторових об'єктів, шляхи найменших втрат, а також швидке накладення растрів для аналізу придатності.

Застосування тріангуляційних даних дозволяє аналізувати поверхні з використанням широкого набору аналітичних функцій. Наприклад, проводити розрахунки об'ємів ділянок до і після проведення земельних робіт; оцінювати області видимості із заданої точки у просторі; визначати висоту, схил та експозицію будь-якої точки на поверхні; створювати профілі висоти на трасі (для доріг або інженерних комунікацій).

Користувач також має визначити, які типи карт слід побудувати за результатами моделювання. Якщо необхідно створити карти з детальним відображенням просторових об'єктів, то слід використати векторні дані. Якщо необхідно створити карти, на яких зображено області зі зрозумілими атрибутивними значеннями, то доцільно застосовувати растрові і тріангуляційні дані.

Запитання для самоперевірки

1. Що містить технічне (апаратне) забезпечення ГІС ?
2. З чого складається програмне забезпечення ГІС ?
3. Які функції додає до СУБД програмне забезпечення ГІС ?
4. У чому полягають переваги і недоліки моделювання поверхонь за допомогою ізоліній ?
5. У чому полягають переваги і недоліки моделювання поверхонь за допомогою растрових даних ?
6. Які функції аналізу поверхонь надає програмне забезпечення стосовно роботи з нерегулярною тріангуляційною мережею (TIN) ?
7. Що має проаналізувати користувач під час вибору географічного подання набору даних ?

Лекція № 3

СТРУКТУРА БАЗИ ГЕОДАНИХ

Навчальні цілі:
- розглянути структури каталогу і бази геоданих;
- вивчити структуру бази геоданих у додатку ArcInfo ArcCatalog.

Навчальні питання:
1. Структура каталогу і з’єднання з геоданими.
2. Структура бази геоданих.

3.1 Структура каталогу і з'єднання з геоданими

Історично склалося, що всі дані зберігаються в комп'ютері у папках, документах, електронних таблицях і базах даних. Для зберігання листів або повідомлень використовують документи; електронні таблиці застосовують для ведення фінансових документів; різноманітні списки клієнтів або каталоги продукції містяться в базах даних. При цьому всі ці дані (файли) розміщують у певному порядку, який ще називають ієрархією папок. Наприклад, упорядкування проводять за клієнтами, проектами, часовим інтервалом та ін.

У сучасних ГІС управління даними також здійснюється за допомогою ієрархії папок, файлів і баз геоданих. У базах геоданих і файлах зберігаються основні типи географічних даних – векторні, растрові, TIN, адреси та локатори (місцеположення). Залежно від розміру БГД географічні дані можуть розміщуватися на диску комп'ютера у персональній БГД або на сервері у багатокористувальній БГД. Бази геоданих і файли створюються за тематикою, структурою установи, типом проекту та іншими принципами.

Дослідження і створення географічних даних, а також забезпечення доступу до даних і управління ними в сучасних ГІС здійснюється за допомогою каталогу. **Каталогом** називають сукупність з'єднань з географічними даними. За допомогою каталогу забезпечується єдине бачення географічних даних, що знаходяться у файлах та персональних БГД. Це досягається використанням відомої деревовидної ієрархії. Каталог також дозволяє проникати у реляційні бази даних і показувати частину їх внутрішньої структури, наприклад, таблиці, в яких зберігаються географічні дані.

Крім того, за допомогою спеціальних піктограм каталог показує структуру географічних даних. Піктограми мають різні позначення, які символізують різні елементи географічних даних. Одна частина піктограм зображує папки і файли, що збігаються з позначеннями у файловій системі Windows. Друга частина являє собою спеціальні піктограми, що

позначають набори просторових і непросторових об'єктів у базах геоданих. Третю частину призначено для з’єднання з базами геоданих та реляційними базами даних, які доступні в інших мережах.

У ГІС ARCGIS (ArcInfo) функції каталогу реалізовано у додатку ArcCatalog, в якому спосіб подання геоданих подібний до того, що використовує провідник Microsoft Windows (рисунок 3.1).

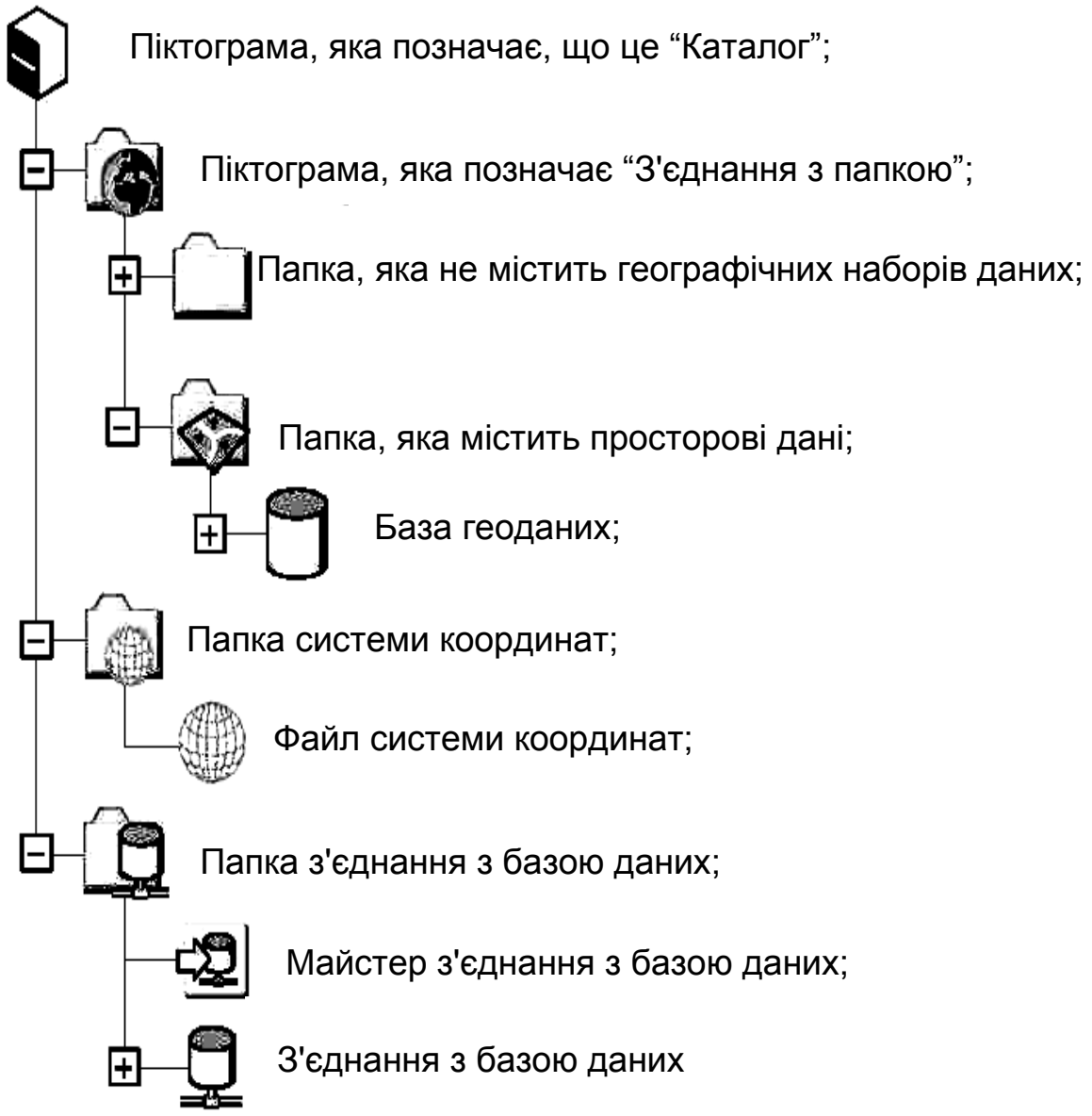


Рисунок 3.1 – Подання каталогу, папок і з'єднань у додатку ArcCatalog

Розглянемо, яка інформація зберігається у каталозі. Як вже було зазначено, “Каталог” містить подання карт і просторових даних, що зберігаються на локальних і мережних дисках. Для того, щоб можна було вказувати особливі місця розміщення даних і підтримувати типи файлів, каталог слід налагодити.

З каталогу можна вибрати піктограму “З'єднання з папкою”, яка вказує на кореневу папку на локальному диску або вибрану папку на мережному диску. З'єднання з папками і базами даних створюють уніфіковане подання всіх даних. З'єднання з папкою дозволяє звертатися до даних на локальних дисках або до дисків на комп'ютерах, підключених до мережі.

У каталозі відображаються всі папки, що знаходяться у складі кожного з'єднання з папкою. Якщо папка не містить географічних наборів даних, вона виглядає як звичайна папка, яку використовує провідник Microsoft Windows. Папки, які містять просторові дані, позначаються спеціальною піктограмою. В них знаходяться бази геоданих, покриття, шейп-файли, файли САПР і супутні файли.

База геоданих усередині папки з просторовими даними позначається спеціальною піктограмою у вигляді циліндру (див. рисунок 3.1) і являє собою об'єктно-орієнтоване сховище просторових даних. Це означає, що ця БГД є персональною базою геоданих. У каталозі також знаходиться папка системи координат, яка містить ряд файлів, що описують системи координат. Файл системи координат позначається спеціальною піктограмою у вигляді шару з координатною сіткою (див. рисунок 3.1) і містить математичний опис перетворення координат на еліпсоїді у координати на площині.

Крім того, у каталозі є папки з'єднань з базами даних, які містять з'єднання з реляційними базами даних і поширеними даними, розміщеними у багатокористувальних реляційних базах даних. Для забезпечення встановлення та перевірки з'єднання з базою даних ARCSDE або базою даних OLE DB всередині цієї папки є майстер з'єднання з базами даних.

Піктограма “З'єднання з базою даних” позначає посилання на базу геоданих, що зберігається у багатокористувальній СУБД і є доступною через ARCSDE, або на негеографічну реляційну БД, яка є доступною через ODBC. З'єднання з базою даних містить параметри для звернення до бази даних: ім'я сервера або IP-адресу, екземпляр або номер порту TCP, ім'я користувача і пароль.

У разі успішного з'єднання з віддаленою багатокористувальною БГД у каталозі розкриються вузли її структури, а на екрані комп’ютера користувача з'явиться таке ж саме дерево об'єктів даних, як і у з'єднаннях з папками. Географічні дані подаються у вигляді ієрархії об'єктів даних за тематикою, робочими групами, загальним просторовим екстентом і системою координат або за топологічними зв'язками.

Нагадаємо, що ARCSDE (SDE – Spatial Database Engine) – це серверне програмне забезпечення у складі ARCGIS Server. ARCSDE призначено для організації зберігання і управління просторовими даними у реляційній СУБД. ARCSDE дозволяє управляти географічною інформацією, що зберігається у таких комерційних реляційних СУБД, як

Oracle, Microsoft SQL Server, IBM DB2 та IBM Informix, а також обслуговувати файли ESRI за допомогою ARCSDE for Coverages.

OLE DB (Object Linking and Embedding, Database) розроблено компанією Microsoft для доступу до різних типів даних, які зберігаються в єдиній формі. OLE DB являє собою набір інтерфейсів, реалізованих за допомогою Component Object Model (COM), які дозволяють додаткам звертатися до даних, що зберігаються у різних джерелах інформації або сховищах даних, за допомогою уніфікованого доступу.

ODBC (Open Database Connectivity) – це програмний інтерфейс доступу до баз даних, який розроблено компанією Microsoft спільно з Simba Technologies.

З використанням каталогу можна виконувати різні операції:

- створювати і форматувати нові дані;
- здійснювати пошук даних;
- оцінювати просторовий екстент і придатність даних виконувати певні завдання;
- документувати походження і якість даних;
- запускати ГІС-операції;
- публікувати дані для широкого використання.

Як вже зазначалося вище, бази геоданих можуть бути двох видів. Персональні БГД використовують файли .mdb (Microsoft Access), а багатокористувальні БГД підтримуються створеним компанією ESRI сервером просторових даних ARCSDE на основі будь-якої з реляційних СУБД. Ці два види БГД є функціонально ідентичними. Однак багатокористувальні БГД підтримують версії даних, що дозволяє багатьом користувачам одночасно звертатися до загальної бази геоданих і її редагувати.

3.2 Структура бази геоданих

Для подання і зберігання інформації у реляційній СУБД ARCGIS використовується об'єктно-реляційна модель даних, яку називають базою геоданих. Ця модель даних дозволяє описувати не тільки геометрію об'єктів, але і їх поведінку, правила взаємозв'язку з іншими класами об'єктів та об'єктами бази геоданих. Один раз описана поведінка об'єктів стає доступною для застосування у всіх додатках ARCGIS Desktop – ArcCatalog, ArcMap, ArcToolBox, а також у ArcObjects – об'єктно-орієнтованих бібліотеках, що розроблені для ARCGIS.

Структурно база геоданих є поєднанням набору географічних даних, класів просторових об'єктів, об'єктних класів і класів відношень.

База геоданих зберігає **набори географічних даних** у вигляді безшовних даних, тобто при такій організації зберігання загальний екстент даних не поділяється на мозаїку самостійних фрагментів (листів). Для безперервного подання екстента у БГД використовується просторове

індексування, що є ефективним способом зберігання даних. Масиви даних малих і середніх розмірів, як правило, зберігаються у персональних базах геоданих. Великі масиви даних зберігаються на сервері підприємства, а їх оброблення може здійснюватися за допомогою ARCSDE.

Клас просторових об'єктів являє собою зібрання просторових об'єктів за одним типом розмірності: точки, лінії або полігону. У свою чергу класи просторових об'єктів поділяють на дві категорії – прості й топологічні класи просторових об'єктів.

Класи простих просторових об'єктів містять точки, лінії, полігони або підписи без яких-небудь топологічних зв'язків між ними. Отже, точка одного класу просторових об'єктів може збігатися з кінцевою точкою лінії іншого класу просторових об'єктів, але це будуть дві різні точки. Такі просторові об'єкти можна редагувати незалежно один від одного.

Класи топологічних просторових об'єктів містять просторові об'єкти, які створюють графи. **Граф** є об'єктом, що пов'язує декілька класів просторових об'єктів в одну інтегровану топологічну одиницю. Одним з типів графів у БГД є геометричні мережі.

Об'єктний клас являє собою таблицю у базі геоданих, з якою можна асоціювати деяку поведінку просторового об’єкта. Об'єктні класи зберігають описову інформацію про об'єкти, які пов'язані з просторовими об'єктами на карті, але самі при цьому не є елементами карти. Наприклад, об'єктним класом може бути список власників земельних ділянок. У цьому випадку можна встановити з'єднання між класом полігональних просторових об'єктів, що являють собою земельні ділянки, і об'єктним класом власників.

Клас відношень – це таблиця, що містить відношення між об'єктами двох класів просторових об’єктів або таблиць. Відношення моделюють залежності між об'єктами. За допомогою відношень можна вказати, що відбудеться з об'єктом, якщо пов'язаний з ним об'єкт буде видалено або змінено.

У додатку ArcCatalog елементи бази геоданих відображено у вигляді спеціальних піктограм, як показано на рисунку 3.2.

База геоданих розташовується у папці з географічними даними і позначається спеціальною піктограмою у вигляді циліндра. Нагадаємо, що база геоданих у цій папці є персональною БГД.

У базі геоданих знаходиться декілька типів географічних даних.

По-перше, **набір растрових даних,** в якому зберігаються зображення або вибірки даних, що розміщуються на регулярній прямокутній сітці. Набір растрових даних може містити один або декілька каналів.

По-друге, **клас точкових просторових об'єктів**, до якого належать прості просторові об'єкти, що мають геометрією точки або мультиточки. Точковими просторовими об'єктами можна, наприклад, позначати місцеположення будівель у місті.

По-третє, **клас лінійних просторових об'єктів**, до якого належать прості просторові об'єкти, що мають геометрію полілінія. Лінійні просторові об'єкти можуть формуватися з використанням трьох видів сегментів: ділянок прямих, дуг кіл і сплайнів Безьє. Лінії можуть використовуватися для відображення меж географічних об'єктів або доріг.

Рисунок 3.2 – Подання бази геоданих у додатку ArcCatalog

По-четверте, **клас полігональних просторових об'єктів**, до якого належать прості просторові об'єкти, що мають геометрію полігону. Полігональні просторові об'єкти зображують області, межі яких можуть бути складені з сегментів – прямої лінії, дуги кола або сплайну Безьє. Полігони можуть мати просту закриту форму або мати дискретні частки.

Полігональні просторові об'єкти можна використовувати для подання географічних об'єктів типу ділянок місцевості, лісових масивів та ін.

База геоданих також містить **об'єктний клас**. Як вже зазначалося вище, цей клас являє собою таблицю, в якій зберігаються об'єкти з заданою поведінкою. Рядки таблиці відповідають об'єктам, а стовпці – атрибутам об'єктів.

Крім того, у БГД є **набір класів об'єктів**, що містить зібрання класів просторових об'єктів, графів і класів відношень, в яких використано загальну систему координат. У цьому наборі також знаходиться декілька типів даних.

Клас з'єднань містить просторові об'єкти, що є простими або складними з'єднаннями, що використовуються під час створення геометричної мережі. Існують прості й складні мережні просторові об'єкти з'єднань.

Просторовий об'єкт простого з'єднання може, наприклад, бути використаним, щоб зобразити естакаду, що пов'язує дві автомобільні дороги.

Просторовий об'єкт складного з'єднання може містити внутрішні частини, які відіграють логічну й топологічну ролі у великій мережі. Наприклад, просторовий об'єкт складного з'єднання можна використовувати, щоб зображати перемикач у мережі електропостачання. В одній позиції перемикач з'єднує лінію електропостачання з одним користувачем, а в іншій – з другим користувачем. При цьому положення перемикача може бути відображено різними символами.

Клас ребер містить просторові об'єкти, що є простими або складними ребрами, що також використовуються під час створення геометричної мережі.

Просторовий об'єкт простих ребер пов'язується з просторовими об'єктами з'єднань в його кінцевих точках. Такий просторовий об'єкт можна використати, щоб зобразити, наприклад, дорогу між двома населеними пунктами у мережі шляхів заданого району. Просторові об'єкти простих ребер відповідають правилам забезпечення зв'язності. Наприклад, зазначена дорога у кінцевій точці має з'єднатися з іншою дорогою в іншому напрямку.

Просторовий об'єкт складного ребра може підтримувати одне або більше з'єднань по всій його довжині, але при цьому він залишається одним просторовим об'єктом. Наприклад, дорога між двома населеними пунктами може мати додаткові з'єднання, тобто, де інші лінії (дороги) приєднуються до неї.

Геометрична мережа визначає набір класів з'єднань і ребер, які разом утворюють лінійну мережу. У базі геоданих просторові об'єкти можна сформувати таким чином, щоб вони створили плоску топологію або геометричні мережі. Топологічно пов'язані ребра й просторові об'єкти з'єднання у межах набору даних можуть обмежувати геометричну мережу.

Це є корисним, коли просторові об'єкти мають бути пов'язані один з одним без розривів. Наприклад, так організовані лінії електропостачання, трансформаторні підстанції, перемикачі у мережі електропостачання.

У наборі класів об'єктів також зберігається **клас відношень**. Як вже зазначалося вище, цей клас являє собою набір відношень між об'єктами двох класів просторових об'єктів.

У плоскій топології у класах просторових об'єктів можна використовувати геометрію, що є спільною з геометрією інших класів просторових об'єктів. Наприклад, можна визначати топологічні відношення між вулицями, кварталами, групами кварталів і ділянками, що обслуговуються будь-яким комунальним підприємством. При цьому вуличні сегменти будуть визначати межу кварталу, який вони утворюють. У свою чергу, групи кварталів можуть бути зібрані в об'єднання груп, а об'єднання груп – у ділянки. Під час модифікації межі одного просторового об'єкта також будуть модифіковані межі, які використовуються разом з нею.

Зазначимо ще раз, що доступ до багатокористувальних БГД здійснюється за допомогою папки з'єднання з базою даних (див. рисунок 3.1). Багатокористувальна БГД містить такі ж самі набори й класи просторових об'єктів, що й персональна БГД (див. рисунок 3.2).

Запитання для самоперевірки

1. Як здійснюється управління даними у сучасних ГІС ?
2. Що таке каталог і що він забезпечує, дозволяє та показує ?
3. Які операції виконуються у каталозі ?
4. Що таке база геоданих і з яких структурних елементів вона складається ?
5. У чому полягає різниця між класами простих і топологічних просторових об'єктів ?
6. Для чого призначено об'єктний клас і що він собою являє ?
7. Для чого призначено набір класів об'єктів та які існують набори класів об'єктів ?

Лекція № 4

СТРУКТУРА ДАНИХ ПОКРИТТЯ І ШЕЙП-ФАЙЛІВ

Навчальні цілі:
- розглянути подання даних покриття і робочої області;
- вивчити структуру шейп-файлів.

Навчальні питання:
1. Подання даних покриття і робочої області.
2. Структура шейп-файлів.

4.1 Подання даних покриття і робочої області

Розроблення й застосування баз геоданих у сучасних ГІС не виключає можливості застосування покриття, що було створено раніше. Введення баз геоданих доповнює й розширює можливості зберігання та подання даних у ГІС. Якщо користувача, що виконує певні завдання, задовольняють можливості подання даних покриття, він може використовувати весь спектр цих можливостей. У нових версіях програмного забезпечення (додатках ГІС ArcInfo) одночасно з базою даних можна відображати, запрошувати, аналізувати і редагувати покриття. У разі необхідності, а також за наявності часових та інших витрат на перетворення можна перевести покриття у бази геоданих. Фактично база геоданих є розвитком моделі покриття, можна сказати, що база геоданих – це покриття наступного покоління.

Покриття об'єднують просторові й атрибутивні дані, а також зберігають топологічні зв'язки між просторовими об'єктами. При цьому просторові дані зберігаються у двійкових файлах, а атрибутивні дані й топологічні зв’язки – у таблицях INFO.

Каталог об'єднує подання двійкових файлів покриття і таблиць INFO у **класи просторових об'єктів покриття**.

Робоча область містить географічні дані у трьох формах:
- у покриттях з векторними даними;
- грідах з растровими даними;
- нерегулярних тріангуляційних мережах (TIN), які зображають поверхні.

Робоча область – це особлива папка (папка особливого типу), в якій зберігаються покриття, гріди та нерегулярні тріангуляційні мережі, а атрибути – у таблицях INFO. Крім того, всі таблиці управляються через підпапку INFO, яка не відображається у каталозі. Цілісність даних при виконанні у каталозі операцій щодо створення, переміщення і видалення елементів робочої області підтримується автоматично. Провідник Windows є не придатним для управління покриттями, грідами або TIN. Це

обумовлено тим, що провідник Windows не забезпечує синхронізацію між покриттями та підпапкою INFO, і в цьому випадку дані будуть зіпсованими. Тому для управління покриттями, грідами або TIN використовують особливу папку – робочу область**.**

Покриття містять класи просторових об'єктів, які складаються з однорідних об'єктів. Існують первинні, похідні (вторинні) та складні (складені) типи просторових об’єктів покриття.

До первинних типів просторових об'єктів покриття належать: **точки, вузли, дуги та полігони** (рисунок 4.1).

Точкові просторові об'єкти можуть являти собою невеликі географічні об'єкти, наприклад, свердловини, колодязі, будівлі та ін. Крім того, точками також маркірують (відзначають) внутрішні області полігонів. Такі точки називають **точкою мітки**. Крім того, точками міток пов'язують атрибути з полігонами. Кожен полігон у покритті має одну точку мітки з номером об'єкта (ID), яка, як правило, розташовується біля центра багатокутника.

Вузли являють собою кінцеві точки та зв'язки між дугами. Вузлами можуть бути точкові об'єкти у будь-якій мережі, а також вони можуть мати атрибути. Дуже часто вузли використовуються для створення (прокладки) шляхів, тому що об'єкти в покриттях топологічно з'єднані між собою.

Дуги являють собою сполучені набори лінійних сегментів з вузлами у кінцевих точках. Окрема дуга може бути автономною, а декілька дуг можуть бути об’єднані у лінійні мережі, наприклад, інженерні мережі. За допомогою дуг також можна організовувати полігони, якими є області, наприклад, поля або ділянки лісу.

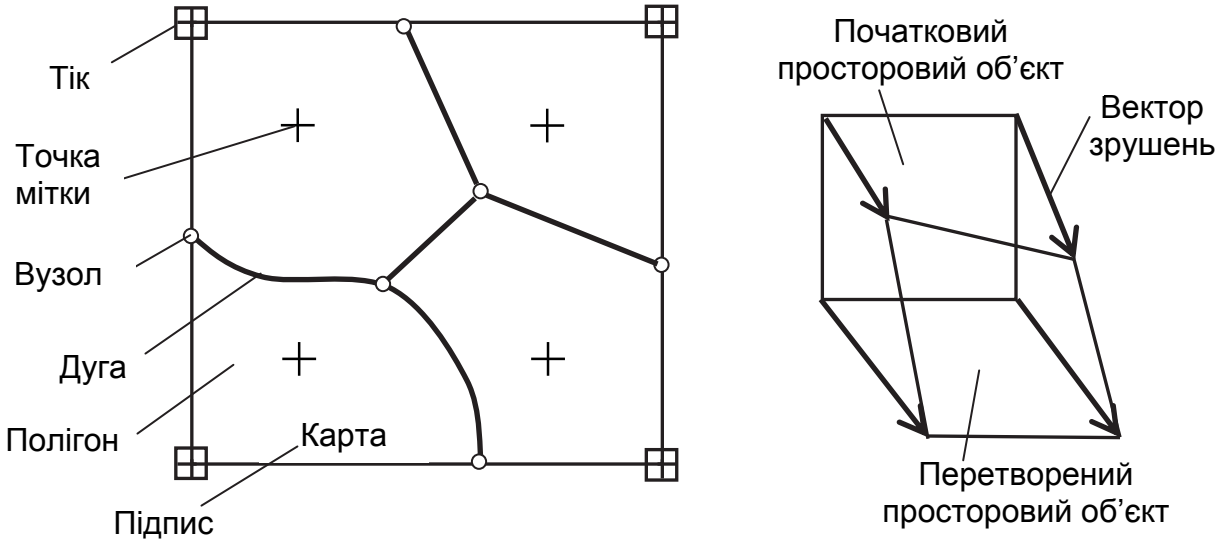


Рисунок 4.1 – Подання первинних і похідних типів просторових об’єктів у покритті

Полігони являють собою області, які обмежені дугами, включаючи дуги, які визначають острівні полігони. Кожна точка області потрапляє точно всередину одного полігону.

Ще раз зазначимо, що первинні просторові об'єкти мають топологічні зв'язки. Наприклад, дуги утворюють периметри полігонів, а вузли є кінцевими точками дуг.

До похідних (вторинних) типів просторових об’єктів покриття належать: **тіки, вектори зрушень і підписи (анотації)** (див. рисунок 4.1).

Тіки являють собою географічні контрольні точки, які використовують для реєстрації карти. Ними позначають відомі місцеположення на землі, а також їх застосовують для перетворення координат покриття. Тики дозволяють точно перетворити оцифровані об'єкти на паперовій карті від одиниць вимірювання дигитайзером у сантиметрах або дюймах до одиниць вимірювання реального миру типу кілометр або миля.

Вектори зрушень використовують для геометричного корегування форми просторових об’єктів покриття, наприклад, для узгодження меж суміжних покрить. Вектори зрушень перетворюють одну точку в іншу.

Підписи (анотації) являють собою текстові рядки, які описують просторовий об'єкт, коли карту відображено або надруковано. Підпис може позиціонуватися в точці, між двома точками або вздовж ряду точок. Підписи використовують для полегшення читання та розуміння карти. Підписи зберігаються у географічних координатах, які визначають їх позицію відносно інших об'єктів покриття і масштаб.

Покриття містять також складні (складені) просторові об'єкти: **маршрути (секції) та регіони** (рисунок 4.2).

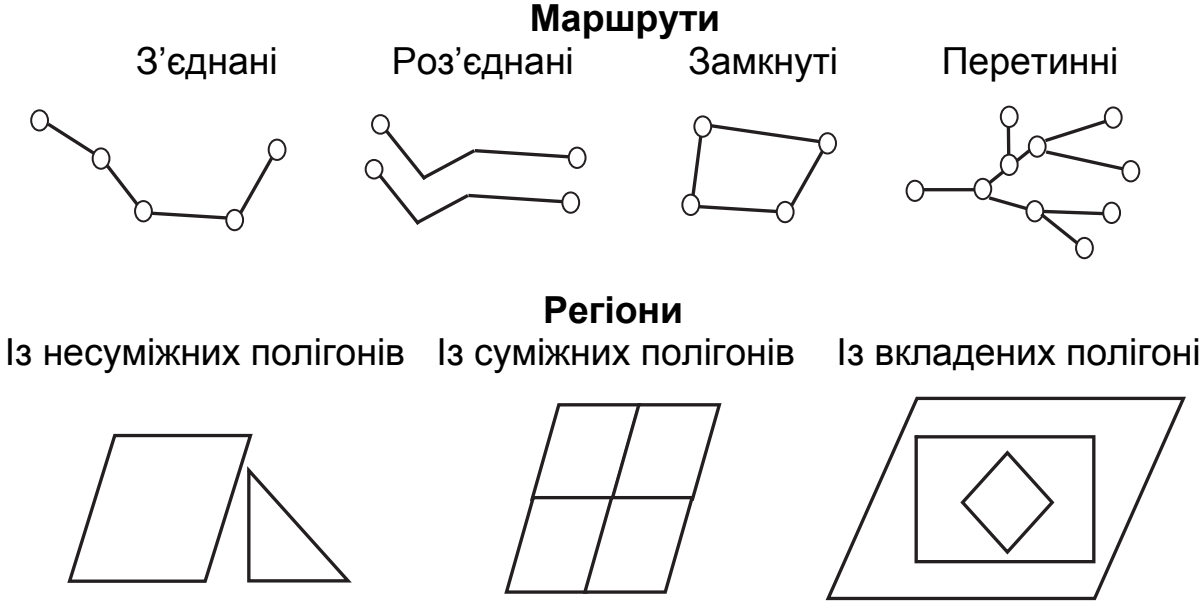


Рисунок 4.2 – Подання складного типу просторових об’єктів у покритті

Маршрути являють собою лінійні просторові об'єкти, які складаються із секцій. У свою чергу секції складаються з дуг або частин дуг. Маршрути можуть бути з'єднаними, роз'єднаними, замкнутими або перетинними.

Маршрути, як правило, використовують для моделювання транспортних систем, тобто маршрути визначають шляхи на існуючій лінійній мережі (мережах доріг, вулиць та ін.). Іноді точки, які становлять інтерес на мережі, не знаходяться у вузлах, тому секції можуть ідентифікувати частини дуг. При цьому секція визначає (фіксує), де починається або закінчується маршрут на даній дузі.

Регіони являють собою площадкові просторові об'єкти, які складаються з набору полігонів (несуміжних, суміжних або вкладених).

Регіони, як правило, використовують для моделювання елементів землекористування та навколишнього середовища. Наприклад, материк і острів можна відобразити як два несуміжних полігони, але вони можуть належати одному регіону. Регіони із суміжних полігонів можна, наприклад, застосовувати для зображення лісу різного типу (хвойного або листяного). Регіон з вкладених полігонів можна використовувати для зображення району пожежі в лісовому масиві.

Покриття також містять гріди та нерегулярні тріангуляційні мережі.

Грід являє собою прямокутну матрицю чарунок, якою є растрові зображення або сіткові вибірки. Атрибути значень чарунок зберігаються в таблиці атрибутів растру (VAT).

Нерегулярні тріангуляційні мережі (TIN) складаються з точок із *z*-значеннями, які з'єднані в мозаїку трикутників для зображення поверхні.

У додатку ArcCatalog елементи покриття й робочої області наведено у вигляді спеціальних піктограм, як показано на рисунку 4.3. Із цього рисунку видно, що у спеціальній папці (робочій області ArcInfo) знаходяться папки з покриттями, грідами й TIN, а також таблиця INFO з атрибутами більшості класів просторових об'єктів.

Як зазначалося вище, покриття – це топологічно інтегровані зібрання класів просторових об'єктів. Точкове покриття містить клас точкових просторових об'єктів, а також може складатися з класів тіків, векторів зсуву та підписів. Лінійне покриття містить клас дуг, а також може складатися з класів вузлів, маршрутів, точок, тіків, векторів зсуву та підписів. Полігональне покриття містить класи полігональних і точкових просторових об'єктів, а також може складатися з класів регіонів, дуг, вузлів, маршрутів, тіків, векторів зсуву та підписів.

На рисунку 4.3 показано піктограму, що позначає покриття з полігональною топологією. Всередині цього покриття знаходяться класи просторових об'єктів покриття. Розглянемо їх більш докладно.

Клас точкових просторових об'єктів покриття містить точкові просторові об'єкти, атрибути яких зберігаються в таблиці атрибутів точок.

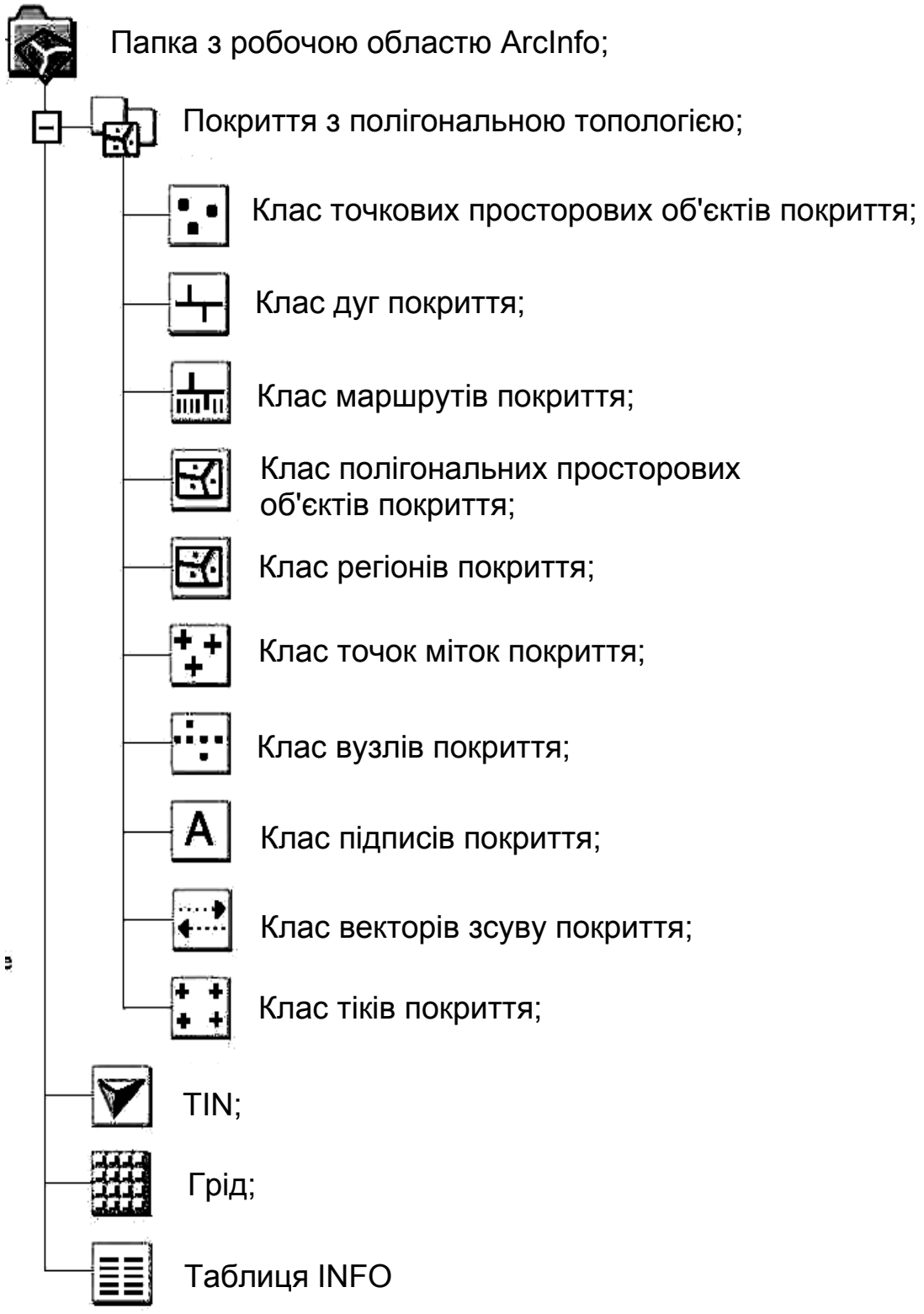


Рисунок 4.3 – Подання покриття і робочої області ArcInfo у додатку ArcCatalog

Клас дуг покриття містить лінійні просторові об'єкти, які створюють мережу або визначають межі полігонів. Атрибути цих об'єктів зберігаються в таблиці атрибутів дуг.

Клас маршрутів покриття складається із складних лінійних просторових об'єктів з лінійною системою координат, атрибути яких зберігаються у таблицях підкласів маршрутів.

Клас полігональних просторових об'єктів покриття містить площадкові просторові об'єкти, що обмежені дугами та позначені внутрішніми точками міток. Атрибути цих об'єктів зберігаються в таблиці атрибутів полігонів.

Клас регіонів покриття містить складні площадкові просторові об'єкти, кожен з яких може утворюватися декількома полігонами. При цьому їх атрибути зберігаються в таблицях підкласів регіонів.

Клас точок міток покриття складається з точок міток, якими позначають полігони. Кожен полігон має одну точку мітки.

Клас вузлів покриття містить вузли, які знаходяться на кінцях дуг. Атрибути вузлів зберігаються у відповідній таблиці атрибутів вузлів.

Клас підписів покриття складається з тексту, що розміщується на карті. Атрибути цих об'єктів можуть зберігатися в таблицях підкласу тексту.

Клас векторів зсуву покриття містить вектори, що застосовують для вирівнювання локальних областей на основі відомих опорних точок. Таке перетворення називають локальною нелінійною трансформацією.

Клас тіків покриття містить точки, які використовують для реєстрації карт під час їх цифрування.

Разом з покриттями у робочій області ArcInfo знаходяться гріди, TIN і таблиця INFO.

TIN складається з точок із z-значеннями, з'єднаними у мозаїку трикутників для тривимірного зображення поверхні.

Грід являє собою прямокутну матрицю чарунок, яка має растрові зображення або сіткові вибірки. Атрибути значень чарунок зберігаються в таблиці атрибутів растру.

Таблиця INFO – це таблиця реляційної бази даних. Таблиці атрибутів просторових об'єктів покриття, які зазначалися вище, є таблицями INFO, що з'єднані з просторовими об'єктами через їх ідентифікатори.

4.2 Структура шейп-файлів

Часто користувачеві не потрібно використовувати складне подання просторових даних на картах, що створюються за допомогою баз геоданих або покриттів. У цьому випадку він може скористатися більш простими формами подання просторових даних, які містяться у шейп-файлах. Шейп-файли також можна використовувати для деяких видів аналізу. Крім того, існує велика кількість географічних даних, які надані у форматі шейп-файлів.

Використовуючи класи простих просторових об'єктів, у шейп-файлі відображають об'єкти за допомогою точок, ліній і полігонів, але ці об'єкти не мають топологічних зв'язків. Перевагами такої структури є простота і

висока швидкість відображення об’єктів, але шейп-файли не забезпечують цілісності просторових даних. Наприклад, на картах земельних ділянок потрібно, щоб полігони, які створюють ділянки, не перекривалися і не мали зазору між собою. Шейп-файли не можуть гарантувати такий тип просторової цілісності. Кожен шейп-файл зберігає об'єкти, що належать одному класу просторових об'єктів, тобто шейп-файл являє собою однорідний набір об'єктів, які можуть мати форму точки, мультиточки, полілінії або полігону (рисунок 4.4).

Шейп-файли мають два типи точкових об'єктів: точки й мультиточки (див. рисунок 4.4).

Точковий шейп-файл містить просторові об'єкти з геометрією «точка». Точка – це одна пара координат. Точкові форми мають прості просторові об'єкти, наприклад, стовпи, колодязі та ін.

Мультиточковий шейп-файл містить просторові об'єкти з геометрією «мультиточка», в якому декілька точок являють собою один просторовий об'єкт. Багатоточкові форми мають складні просторові об'єкти, наприклад, група маленьких островів, група нафтових свердловин та ін.

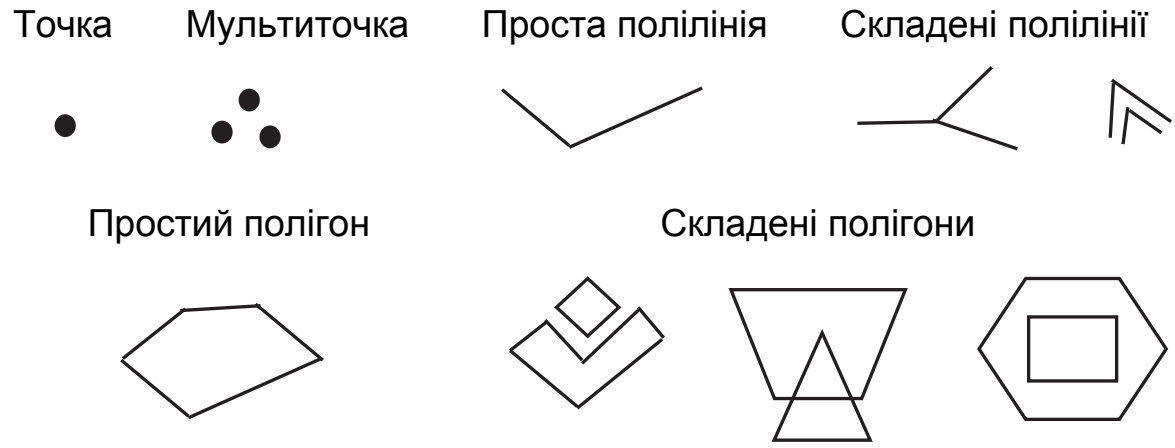


Рисунок 4.4 – Подання просторових об'єктів у шейп-файлах

Шейп-файли містять лінійні просторові об'єкти, які можуть бути простими або складеними полілиніями (див. рисунок 4.4).

Полілінії утворюються з одного або декількох ланцюжків з’єднаних між собою відрізків прямих. Ланцюжки в полілінії можуть бути несуміжними, перетинними, ламаними лініями з відгалуженням і сполученими. Полінії також можуть бути безперервними і мати переривчасті частини. За допомогою поліліній можна відображати, наприклад, річки, транспортні мережі та ін.

Шейп-файли також містять площадкові об'єкти, які складаються з простих або складених полігонів (див. рисунок 4.4).

Полігон може складатися з одного або декількох кілець. Кільцем є замкнутий ланцюжок, який не може перетинати сам себе. Кільця одного полігону можуть бути несуміжними, перетинними, а також вкладеними

один в один. За допомогою полігонів можна зображати, наприклад, лісові масиви, земельні ділянки та ін.

Шейп-файли зберігають атрибути у файлах формату dBASE (.dbf), до яких можна приєднати за допомогою атрибутивного ключа атрибути об'єктів з іншої таблиці dBASE.

У додатку ArcCatalog шейп-файли наводять у вигляді спеціальних піктограм, як показано на рисунку 4.5. У спеціальній папці шейп-файли подано як автономні класи об'єктів: точкових або мультиточкових, лінійних і полігональних. У цій папці також міститься “Таблиця dBASE” з атрибутивною інформацією та “Креслення САПР” з наборами даних САПР, які можна нанести на карту, але не у вигляді шару.

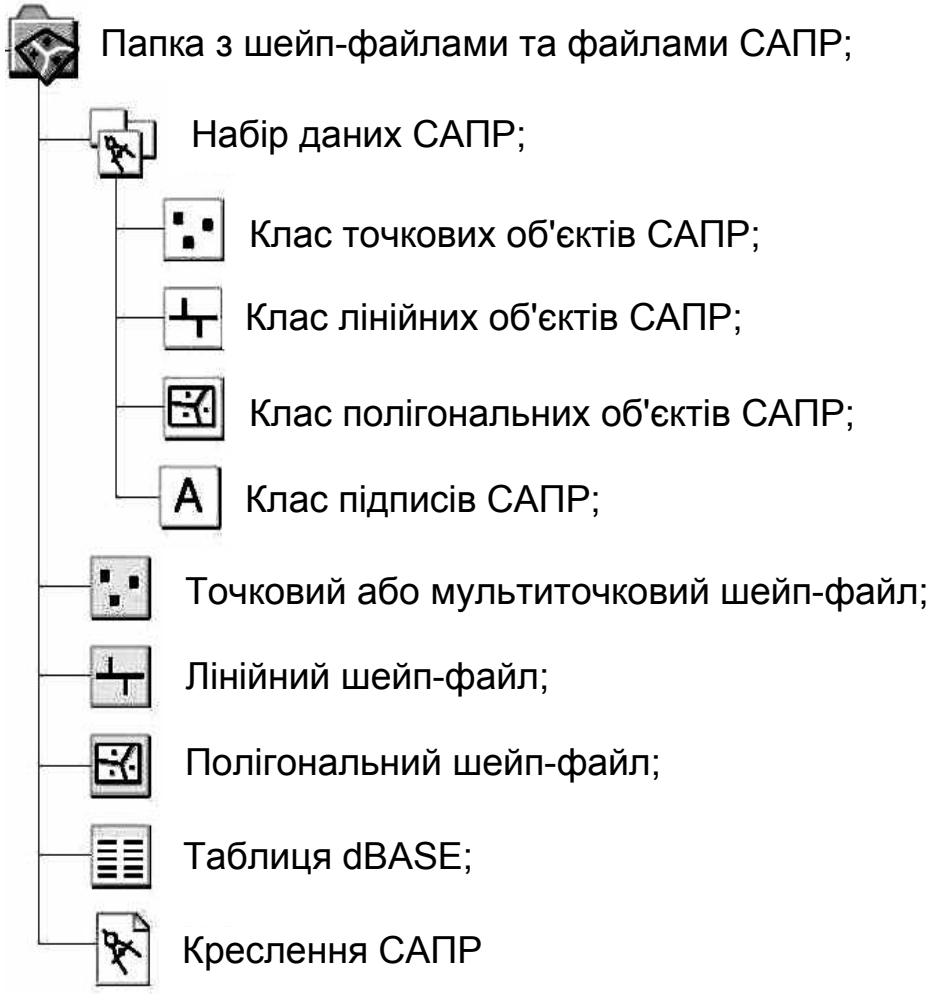


Рисунок 4.5 – Подання шейп-файлів і креслень САПР у додатку ArcCatalog

Крім того, у папці з шейп-файлами можуть зберігатися класи простих просторових об'єктів систем автоматичного проектування (САПР) у форматах AUTOCAD і MicroStation, які є найбільш поширеними сховищами простих просторових об'єктів.

Набір даних САПР містить класи точок, ліній, полігонів і підписів. Набір даних САПР може бути у форматі AUTOCAD або MicroStation.

Клас точкових об'єктів САПР являє собою сукупність шарів САПР всередині набору даних САПР, що містить точки.

Клас лінійних об'єктів САПР – це сукупність шарів САПР всередині набору даних САПР, що містить лінії.

Клас полігональних об'єктів САПР – це сукупність шарів САПР всередині набору даних САПР, що містить полігони.

Клас підписів САПР являє собою сукупність шарів САПР всередині набору даних САПР, що містить текст.

Запитання для самоперевірки

1. Що являє собою робоча область ArcInfo та для чого її призначено ?
2. Які існують типи просторових об’єктів покриття ?
3. Які існують первинні типи просторових об'єктів покриття та для чого їх використовують ?
4. Які існують складні типи просторових об'єктів покриття та для чого їх застосовують ?
5. У чому полягає різниця між шейп-файлами, базами геоданих і покриттями при зображенні просторових об’єктів ?
6. Як подаються прості просторові об'єкти у шейп-файлі ?
7. Для чого призначено набір даних САПР і що він містить ?

Лекція № 5

СТРУКТУРА КАРТ І ШАРІВ

Навчальні цілі:
- вивчити подання карт і шарів;
- провести порівняння структур наборів даних.

Навчальні питання:
1. Подання карт і шарів.
2. Порівняння структур наборів даних.

5.1 Подання карт і шарів

У сучасних ГІС існують широкі можливості щодо створення високоякісних карт. Під час розроблення карт та їх шарів не потрібно програмувати або писати які-небудь макроси, все визначається програмним забезпеченням ГІС. Такий підхід до створення карт, які використовують у різних організаціях, дозволяє стандартизувати їх формат, зміст і зовнішній вигляд.

За допомогою каталогу можна не тільки здійснювати доступ до географічних даних, але й управляти файлами, в яких постійно зберігається карта, а також можна визначати шари карти з усіма заданими для них картографічними специфікаціями.

Карта, що створена за допомогою додатка Arcmap, зберігається на диску комп'ютера у вигляді файла з розширенням .mxd, який називається **документом карти**. Цей документ зберігає елементи оформлення карти, але не містить безпосередньо географічних даних. Для відображення географічних даних у шарах карти створюються посилання на набори географічних даних, що можуть бути розташовані у будь-якому місці на диску комп’ютера або у базі даних, яка є доступною через каталог.

Початковим етапом створення будь-якої карти є вибір існуючого у програмному забезпеченні **шаблону документа карти**. Шаблони значно полегшують процес створення серії карт з єдиним зовнішнім виглядом. Шаблон документа карти може бути достатньо простим, з вказівкою тільки розміру сторінки й стилю, а також може бути складним і містити велику кількість заздалегідь заданих картографічних елементів і шарів.

Під час створення карт користувач застосовує різноманітні **стилі**. Стиль являє собою набір палітр картографічних об'єктів, які можна застосовувати для створення карт. Такі картографічні об'єкти містять маркери, які використовують для відображення точкових просторових об'єктів, лінійні знаки для зображення лінійних просторових об'єктів, заповнення для відображення полігональних просторових об'єктів і текстові символи для нанесення підписів.

Крім того, до об'єктів стилю належать кольори і деякі картографічні елементи, наприклад, покажчики напряму на північ. Стилі забезпечують однакове зображення картографічних умовних знаків (символів) на різних картах. Користувач може самостійно створювати велику кількість стилів для виробництва різних типів картографічних продуктів.

Сучасні ГІС дозволяють користувачеві поділяти інформацію на карті за допомогою логічних категорій, які називають **картографічними шарами**. Вся картографічна інформація у ГІС сформована у вигляді шарів. Електронна карта складається з впорядкованої сукупності графічних шарів карти, які послідовно відображаються на екрані комп'ютера (рисунок 5.1). Шари зазвичай містять інформацію тільки про один тип об'єктів, наприклад, річки і озера, або про невелику групу пов'язаних об'єктів, наприклад, дороги, будинки, телефонні, електричні та інші мережі. У разі потреби користувач може тимчасово підключати необхідні шари або відключати шари, що заважають перегляду. Можна також міняти порядок відображення шарів.

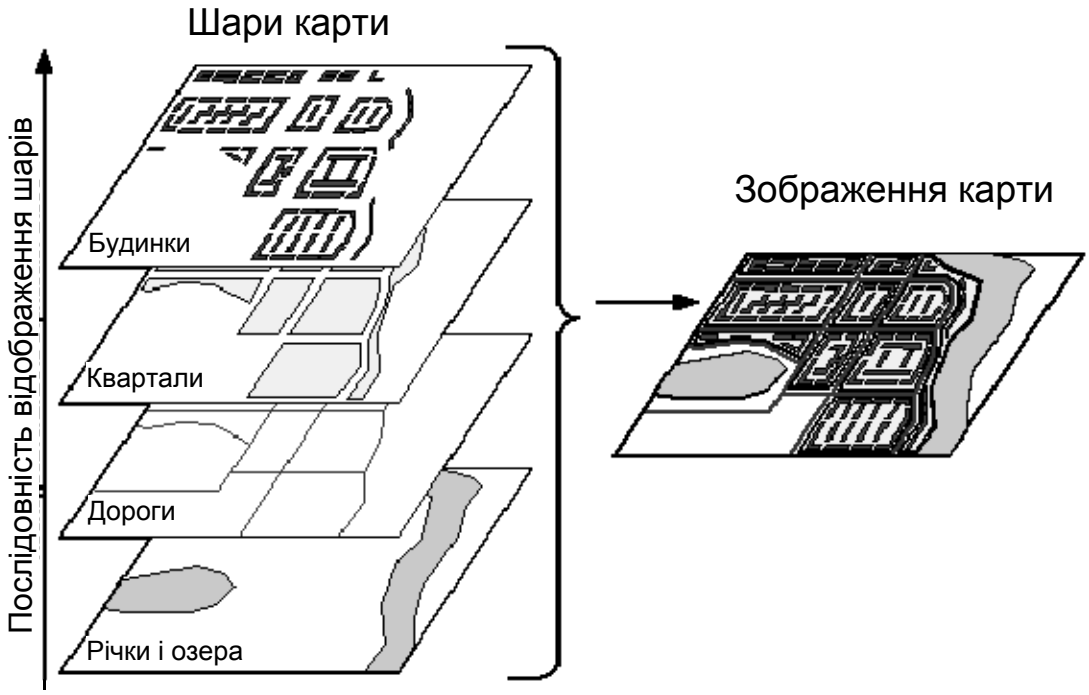


Рисунок 5.1 – Шари карти [5]

Таким чином, шар являє собою сукупність однотипних просторових об'єктів, що належать до одного класу об'єктів у межах деякої території і у системі координат, що є загальною для набору шарів.

Користувач може зберігати шари безпосередньо у самому документі карти або в окремих файлах з розширенням .lyr. Під час побудови простих карт доцільно створювати шари всередині цих карт. Якщо шарами будуть користуватися інші люди, то доцільно їх створювати в окремих файлах. Кожен шар містить посилання на географічні дані, тому, якщо у каталозі

переміщається або перейменовується набір географічних даних, то необхідно корегувати посилання у шарі, щоб всі карти, куди входить цей шар, могли мати ці дані.

Шари бувають точковими, лінійними, полігональними.

Точкові шари містять об'єкти, які можна відобразити у вигляді точок, наприклад, свердловини, колодязі та ін. Точкові шари можуть мати посилання на точки та з'єднання у базі геоданих; точки міток, тіки та вузли у покритті; точки у шейп-файлі або у наборі даних САПР.

Лінійні шари містять об'єкти, які можна зобразити у вигляді ламаної або гладкої лінії, наприклад, річки, дороги та ін. Лінійні шари можуть мати посилання на лінії та ребра у базі геоданих; дуги і маршрути у покритті; лінії у шейп-файлі або у наборі даних САПР.

Полігональні шари містять об'єкти, які можна подати у вигляді обмеженої ділянки, наприклад, лісової ділянки та ін. Остання точка полігону завжди має збігатися з першою точкою, тобто полігон не може мати менше чотирьох точок, а також не може мати самоперетинів. Полігональні шари можуть мати посилання на полігони у базі геоданих; полігони та регіони у покритті; полігони у шейп-файлі або у наборі даних САПР.

Шари підписів можуть мати посилання на текст у базі геоданих або покритті.

Растрові шари можуть мати посилання на гріди у робочій області ArcInfo та файли зображень у різних форматах.

TIN-шари можуть мати посилання на TIN у робочій області ArcInfo.

Файли карт і шарів можуть знаходитися в будь-якому місці в комп'ютері або Інтернеті. Їх можна поміщати в папки з географічними даними або у будь-які інші папки. У каталозі є видимими тільки ті шари, які зберігаються в окремих файлах. Щоб побачити шари, які внесені в документ карти, необхідно звернутися до таблиці змісту карти у додатку Arcmap.

У додатку ArcCatalog карти і шари подають у вигляді спеціальних піктограм, як показано на рисунку 5.2, з якого видно, що у спеціальній папці знаходяться “Документ карти” та “Шаблон документа карти”. Документ карти містить картографічні елементи, які створюють карту: назву карти, покажчик на північ, легенду, масштабну лінійку, фрейми даних і шари. Шаблон документа карти містить задані заздалегідь картографічні елементи для дотримання картографічних стандартів. Документи карт створюють з шаблонів. Крім того, з рисунку 5.2 видно, що у спеціальній папці знаходяться різноманітні шари.

Груповий шар є набором шарів, які об'єднані з метою спрощення їх вставки у карту.

Точковий шар містить посилання на клас нульмірних просторових об'єктів у базі геоданих, покритті, шейп-файлі, кресленні САПР або іншому наборі даних.

Рисунок 5.2 – Подання карт і шарів у додатку ArcCatalog

Лінійний шар містить посилання на клас одновимірних просторових об'єктів: клас лінійних просторових об'єктів або клас ребер у базі геоданих, клас дуг у покритті або лінійний шейп-файл.

Полігональний шар містить посилання на клас двовимірних просторових об'єктів: клас полігональних просторових об'єктів у базі геоданих або покритті, полігональний шейп-файл.

Шар підписів містить посилання на клас підписів у базі геоданих, покритті або наборі даних САПР.

TIN-шар містить посилання на набір даних TIN у базі геоданих або робочій області ArcInfo.

Растровий шар містить посилання на растр у базі геоданих, грід у робочій області ArcInfo або файл зображення у папці.

Шар креслення САПР містить посилання на креслення САПР у будь-якому загальноприйнятому форматі.

5.2 Порівняння структур наборів даних

Сучасні ГІС дають можливість працювати з наборами даних трьох видів: базами геоданих; покриттями; шейп-файлами.

Об'єкти, що знаходяться у цих наборах даних структуруються по-різному. Це обумовлено, перш за все, різними способами зберігання (у папках та базах даних), а також різним поданням топологічної інформації.

Коротко розглянемо як зберігаються дані у цих трьох наборах.

Так, **база геоданих** являє собою об’єднання наборів класів об'єктів, растрів і нерегулярних тріангуляційних мереж (TIN). У БГД усі просторові, топологічні та атрибутивні дані зберігаються в таблиці реляційної бази даних. Бази геоданих покривають будь-які географічні екстенти без поділу. Поведінка даних тісно пов'язана з просторовими об'єктами за допомогою правил і кодів, які написані для призначених для користувача класів просторових об'єктів.

Покриття розміщується в робочій області ArcInfo разом з грідами й TIN. У покритті просторові дані зберігаються у двійкових файлах, а топологічні й атрибутивні дані – у таблицях INFO. У разі великих наборів даних покриття розподіляються на листи в бібліотеці карт. Поведінка даних слабо пов'язана з просторовими об'єктами за допомогою файлів мови макросів AML або макросів VBA (Visual Basic for Applications). Нагадаємо, що VBA є спрощеною версією мови програмування Visual Basic, яку підтримує додаток Microsoft Office.

Шейп-файли зберігаються у папці шейп-файлів. При цьому просторові дані – у двійкових файлах, а атрибутивні дані – у таблицях dBASE. Шейп-файли не містять топологічних даних. Шейп-файли зберігають без поділу набори даних малих і середніх розмірів. Управління просторовими об'єктами здійснюють за допомогою макросів VBA.

Наведені набори даних містять різні набори класів об’єктів. Розглянемо їх коротко.

База геоданих містить класи простих або топологічних просторових об'єктів. При цьому лінійна топологія реалізується за допомогою геометричної мережі, а полігональна – за допомогою топологічного редагування. У БГД можна топологічно поєднувати просторові об'єкти великої кількості класів. Користувачі можуть встановлювати зв'язки між просторовими об'єктами різних класів. Набори класів об’єктів у БГД мають певну систему координат.

Покриття складається з класів топологічних просторових об'єктів, що беруть участь у лінійній або полігональній топології. У покритті лінійна топологія може бути реалізованою за допомогою дуг, вузлів і маршрутів, а полігональна – за допомогою дуг, точок міток, полігонів і регіонів. У покритті можна топологічно поєднувати просторові об'єкти в межах одного класу. При цьому зв'язки підтримуються між залежними просторовими об'єктами,

наприклад, дугами і полігонами. Як і у БГД набори класів об’єктів у покритті мають певну систему координат.

Шейп-файл містить один клас простих просторових об'єктів. Полігональну топологію у шейп-файлі можна реалізовувати за допомогою топологічного редагування. У шейп-файлах зв'язки між просторовими об'єктами не підтримуються. Шейп-файли не мають певної системи координат.

Коротко розглянемо об’єкти, які знаходяться у класі просторових об'єктів у трьох видах наборів даних.

У **базі геоданих** клас просторових об'єктів зберігає просторові об'єкти у реляційній таблиці зі спеціальним полем геометрії об'єкта. Типами просторових об'єктів є точка, лінія, полігон, просте з'єднання, складне з'єднання, просте ребро, складне ребро. У БГД передбачена можливість створення призначених для користувача класів просторових об'єктів.

У **покритті** клас просторових об'єктів містить геометрію просторового об'єкта у двійковому файлі, а атрибути й топологію – у таблиці атрибутів. Первинними типами просторових об'єктів є точка, дуга, полігон і вузол, вторинними типами – тіки, вектори зсуву та підписи, складеними типами – регіон і маршрут. У покритті класи просторових об'єктів не можуть бути розширеними.

У **шейп-файлі** клас просторових об'єктів зберігає геометрію просторового об'єкта у двійковому файлі, а атрибути – у файлі dBASE. Типами просторових об'єктів є точка, мультиточка, лінія та полігон. Шейп-файл не може бути розширеним.

Коротко розглянемо об’єкти, які знаходяться у класі точкових просторових об'єктів у трьох видах наборів даних.

У **базі геоданих** в класі точкових просторових об'єктів містяться точки або мультиточки. Мультиточка являє собою один просторовий об'єкт, що складається з декількох точок. З'єднання у мережах також є точками.

У **покритті** в класі точкових просторових об'єктів розміщено точки, точки міток, вузли й тіки. Точки міток є точковими або центроїдами полігонів з атрибутами. Вузли – це кінцеві точки дуг. Тіки використовують для реєстрації покриття у системі координат. У покритті немає мультиточок.

У **шейп-файлі** в класі точкових просторових об'єктів знаходяться точки й мультиточки. Точки не асоціюються з полігонами.

Коротко розглянемо об’єкти, які зберігаються в класі лінійних просторових об'єктів у трьох видах наборів даних.

У **базі геоданих** клас лінійних просторових об'єктів містить полілінії, які можуть складатися з одного або декількох шляхів. Шляхи мають сегменти чотирьох типів: лінію дуги, коло дуги, дугу еліпса, криву Безьє. У БГД також зберігаються геометричні мережі, які складаються із з’єднань і ребер, що створюють одномірну мережу.

У **покритті** клас лінійних просторових об'єктів містить дуги і секції. Дуги – це ланцюжки з’єднаних прямих із вузлами у кінцевих точках. Секція являє собою повну дугу або частину дуги. З’єднування маршруту може бути довільним.

У **шейп-файлі** клас лінійних просторових об'єктів містить полілінії, які складаються з одного або декількох шляхів. У шейп-файлі немає топологічних зв'язків.

Коротко розглянемо об’єкти, які зберігаються у класі полігональних просторових об'єктів у трьох видах наборів даних.

У **базі геоданих** в класі полігональних просторових об'єктів знаходяться полігони, які складаються з одного або декількох кілець. **Кільце** – це замкнутий шлях без самоперетинів. Полігони можуть складатися з одного кільця, декількох суміжних кілець або вкладених кілець.

У **покритті** в класі полігональних просторових об'єктів розташовано плоский граф з простими полігонами і регіони. При цьому кожен полігон має точку мітки, з якою асоційовані атрибути. Плоский граф – це суцільне покриття області суміжними полігонами, що не перетинаються. Кожна точка в межах цієї області належить одному полігону. Регіон – це складений об’єкт із полігональних просторових об'єктів.

У **шейп-файлі** в класі полігональних просторових об'єктів знаходяться полігони. Полігони у шейп-файлах є такими, що й полігони у БГД, але сегменти у шейп-файлі можуть бути тільки прямими лініями. Полігони можуть складатися з одного кільця, декількох суміжних кілець або вкладених кілець.

Запитання для самоперевірки

1. Які елементи зберігає “Документ карти” ?
2. Для чого призначено “Шаблон документа карти” ?
3. Для чого існує стиль та які є об'єкти стилю ?
4. Що таке шар і для чого його призначено ?
5. Які існують типи шарів ?
6. Де розміщуються файли карт і шарів ?
7. У чому полягає різниця зберігання даних у БГД, покритті й шейп-файлі ?

Лекція № 6

ОСНОВНІ ВИМОГИ ДО БАЗ ДАНИХ

Навчальні цілі:
- розглянути особливості побудови баз даних і файлових систем;
- вивчити основні функції системи управління базою даних (СУБД).

Навчальні питання:
1. Особливості побудови баз даних і файлових систем.
2. Основні функції СУБД.

6.1 Особливості побудови баз даних і файлових систем

На початковому етапі розвитку інформаційних систем для подання наборів даних використовувалися файлові системи. Основною складовою частиною таких систем є файл, що являє собою прикладну програму. Файл – область зовнішньої пам'яті, яка має найменування, в яку можна записувати дані і з якої можна зчитувати ці дані. Управління файлами здійснюється за допомогою спеціальної системи управління файлами. Така система визначає правила присвоєння імені файлу, спосіб доступу до даних, що зберігаються у файлі, а також структуру цих даних. Крім того, система управління файлами забезпечує розподіл зовнішньої пам'яті, відображення імен файлів у відповідних адресах зовнішньої пам'яті, а також доступ до даних. Структура файлів також визначається способом зберігання даних у зовнішній пам’яті.

У більшості існуючих комп'ютерних систем основними пристроями зовнішньої пам'яті є магнітні диски з рухомими головками. Такі магнітні диски призначені для зберігання файлів і конструктивно складаються з набору магнітних пластин (пакетів магнітних поверхонь). Всередині між магнітними пластинами на одному важелі дискретно рухається пакет магнітних головок. Кожному положенню пакета головок (кроку руху пакета головок) логічно відповідає циліндр магнітного диска. На кожній магнітній поверхні циліндр визначає так звану доріжку, в результаті чого кожна магнітна поверхня містить кількість доріжок, що дорівнює числу циліндрів. При розмічуванні магнітного диска (форматуванні диска) кожну доріжку поділяють на однакову кількість блоків. У кожний блок може бути записано максимальну однакову кількість байтів.

Таким чином, для того, щоб записати або зчитати дані з магнітного диска, потрібно вказати номер циліндра, номер магнітної поверхні, номер блока на відповідній доріжці та кількість байтів від початку цього блока, в якому містяться ці дані.

У сучасних файлових системах обмін даними у магнітних дисках порціями, що є менше об'єму блока, не використовується. Це

обумовлюється насамперед відсутністю виграшу в загальному часі обміну в результаті запису або зчитування даних тільки з частини блока. Основний час витрачається на підведення головок до потрібного циліндра та пошук на доріжці потрібного блока, а не на обмін даними з цим блоком. Для того, щоб працювати з частинами блоків, файлова система також має забезпечити буфери оперативної пам'яті відповідного розміру, однак це призводить до ускладнення розподілу оперативної пам'яті.

У результаті цього у всіх файлових системах явно або неявно виділяється деякий базовий рівень, що забезпечує роботу з файлами. Цей базовий рівень відображає набір блоків, що спрямовуються в адресний простір файла. Розмір цих логічних блоків файла збігається з розмірами фізичного блока диска або є кратним йому та, як правило, дорівнює розміру сторінки віртуальної пам'яті, яка підтримується апаратурою комп'ютера спільно з операційною системою.

Вище зазначалося, що система управління файлами визначає правила присвоєння імені файлу. Коротко розглянемо способи іменування файлів. У сучасних файлових системах підтримується багаторівневе іменування файлів. Таке рішення реалізується за допомогою підтримки у зовнішній пам'яті додаткових спеціально структурованих файлів, які називаються каталогами. Під час такої організації повне ім'я файла складається з низки (списку) назв каталогів, а також імені файла у каталозі, що безпосередньо містить даний файл.

У деяких системах управління кожний архів файлів цілком розташовується в одному дисковому пакеті. В цьому випадку повне ім'я файла починається з назви дискового пристрою, на якому встановлено відповідний диск. Таку організацію називають підтримкою ізольованих файлових систем. У деяких файлових системах вся сукупність каталогів і файлів подається як єдине дерево. При цьому повне ім'я файла починається з назви кореневого каталогу, і нема потреби встановлювати на дисковий пристрій які-небудь конкретні диски. Сама система виконує пошук файла за його іменем і запрошує встановлення необхідних дисків. Таку файлову систему називають повністю централізованою.

У сучасних файлових системах, наприклад, ОС UNIX, реалізовано комбіноване рішення. На базовому рівні у цих файлових системах підтримуються ізольовані архіви файлів, один з яких є кореневою файловою системою. Після запуску системи можна об’єднати кореневу файлову систему і ряд ізольованих файлових систем в одну загальну файлову систему. Після створення загальної файлової системи іменування файлів проводиться так само, як при централізованій файловій системі.

Оскільки файлові системи є загальним сховищем файлів, які можуть бути потрібними багатьом користувачам, вони мають забезпечувати й захист файлів. Для вирішення цього завдання використовується авторизація доступу до файлів, яка полягає в тому, що кожному

зареєстрованому користувачеві певної обчислювальної системи указуються дії, які йому дозволені або заборонені під час роботи з кожним існуючим у цій системі файлом. При реалізації такого підходу кожному зареєстрованому користувачеві відповідає пара цілочислових ідентифікаторів, а саме: ідентифікатор групи, до якої належить цей користувач, і його власний ідентифікатор у групі. При цьому кожний файл зберігає повний ідентифікатор користувача, який створив цей файл і визначає, які дії з файлом може виконувати він сам, які є доступними іншим користувачам тієї ж групи і що можуть робити з файлом користувачі інших груп.

Крім того, система управління файлами має забезпечувати режим багатокористувального доступу. Якщо операційна система підтримує багатокористувальний режим, можуть виникати ситуації, коли два або більше користувачів будуть одночасно намагатися працювати з одним і тим же файлом. Труднощів не виникає, коли користувачі збираються тільки прочитати файл, але якщо хоча б один з них буде намагатися змінити файл, то для коректної роботи цієї групи потрібна взаємна синхронізація.

Для забезпечення коректної роботи у файлових системах застосовується певний підхід. Під час виконання операції відкриття файла крім інших параметрів вказується режим роботи (зчитування або заміна). Якщо до моменту виконання операції, що задає деяка програма **A**, файл вже знаходився у відкритому стані, що було задано деякою іншою програмою **B**, причому існуючий режим відкриття був несумісним з бажаним режимом (сумісними є тільки режими зчитування), то залежно від особливостей системи програмі **A** або повідомляється про неможливість відкриття файла у бажаному режимі, або вона блокується доти, доки програма **B** не виконає операцію закриття файла. Крім того, існує можливість синхронізації декількох процесів, що паралельно модифікують один і той же файл. Для цього введено спеціальний механізм синхронізаційних захоплень діапазонів адрес відкритого файла.

Розглянуті вище особливості побудови файлових систем обумовлюють характерні області застосування файлів. Основною областю є зберігання текстових даних: документів, текстів програм та ін. Текстові файли, як правило, створюються і модифікуються за допомогою різних текстових редакторів. Структурно текстові файли являють собою або послідовність записів, що містять рядки тексту, або послідовність байтів, серед яких зустрічаються спеціальні символи (наприклад, символи кінця рядка), тобто структура текстових файлів є достатньо простою.

Файли також використовують для зберігання об'єктних модулів, що являють собою файли з текстами програм. Такі файли створюються за допомогою компіляторів, на вхід яких надходять текстові файли з програмами. Об'єктні файли, як і текстові, мають просту структуру у вигляді послідовності записів або байтів. Різні системи програмування

створюють специфічні для них структури об'єктних модулів. Логічна структура об'єктного модуля є невідомою файловій системі, його структура підтримується системами програмування.

Крім того, існують файли, що створюються редакторами зв'язків і містять образи виконуваних програм. Логічна структура таких файлів, як і об’єктних файлів, є відомою тільки індивідуальному редактору зв'язків і завантажувачу, що являють собою програми операційної системи. Аналогічну структуру мають файли, що містять графічну і звукову інформацію.

Таким чином, файлові системи забезпечують зберігання слабо структурованої інформації, залишаючи подальшу структуризацію прикладним програмам. Однак для вирішення завдань в інформаційних системах файли не завжди бувають зручними.

Інформаційні системи в основному орієнтовано на зберігання, вибір і модифікацію постійно існуючої інформації, тобто даних, які постійно знаходяться на запам’ятовуючому пристрої. При цьому структура інформації може бути дуже складною, і хоча структури даних є різними в різних інформаційних системах, між ними часто буває багато загального.

На початку розвитку обчислювальної техніки проблеми структуризації даних вирішувалися індивідуально в кожній інформаційній системі. Для цього створювалися необхідні надбудови над файловими системами (бібліотеки програм) подібно до того, як це робиться у компіляторах або редакторах. Ці додаткові індивідуальні засоби управління даними, як правило, складали більшу частину інформаційних систем. Крім того, вони практично повторювалися з однієї системи в іншу. В подальшому для управління складно структурованими даними розробники почали відокремлювати цю загальну частину інформаційних систем у систему управління базами даних (СУБД).

Розглянемо, як можна створити систему управління даними з використанням стандартної базової файлової системи.

Наприклад, необхідно створити просту інформаційну систему, яка містить дані про міста світу. Система має виконувати такі дії: видавати список міст, підтримувати можливість зміни даних про кількість населення і площу, поповнення списку. Для кожного міста має видаватися інформація про країну, на території якої воно знаходиться, столицю, географічне положення та ін.

Побудуємо таку інформаційну систему на базі файлової системи. При цьому будемо використовувати один файл і розширимо базові можливості файлової системи за рахунок спеціальної бібліотеки функцій.

У такій системі мінімальною інформаційною одиницею є місто, тобто доцільно, щоб у цьому файлі знаходився один запис щодо кожного міста. При цьому запис має містити такі поля: повну назву міста (**name**); країни, на території якої воно знаходиться (**country**); це місто є столицею або ні (так або ні) (**capital**); кількість населення (**population**); географічну довготу

й широту (**dol, shir**); кількість населення області, якщо це обласний центр (**pop_urb**). У цьому записі мають бути наведені дані про країну, в якій знаходиться це місто: кількість населення (**pop_cntry**); площу (**sqkm_cntry**); географічні координати (**dol_cntry**, **shir_cntry**).

Функціонально інформаційна система має забезпечити можливість багатоключового доступу до цього файла за допомогою унікальних ключів **name**, не дубльованих у різних записах. Крім того, їй слід забезпечити можливість вибору всіх записів із загальним значенням **country**, тобто забезпечити можливість доступу за допомогою неунікального ключа.

Таким чином, створення інформаційної системи на базі файлової системи потребує формування достатньо складної надбудови для багатоключового доступу до файлів. Крім того, буде необхідною надмірність зберігання даних, для кожного міста однієї країни слід повторювати дані про цю країну, а також виконувати складну масову вибірку та обчислення для отримання сумарної інформації про міста та ін.

Для спрощення створення інформаційної системи можна реалізувати підтримання двох багатоключових файлів: міста і країни. Перший файл буде містити поля: **name**, **country**, **capital**, **population**, **dol**, **shir**, **pop_urb**, а другий – **cntry_name, pop_cntry, sqkm_cntry, dol_cntry, shir_cntry**. При цьому кожен з цих файлів буде містити тільки недубльовану інформацію через те, що немає необхідності виконувати динамічні обчислення сумарної інформації. Однак при такій побудові необхідно передбачити, щоб інформаційна система працювала з двома інформаційно пов'язаними файлами. Ця функція більш властива базі даних, а не файловій системі. Крім того, необхідно забезпечити, щоб інформаційна система розрізняла структуру та сенс кожного поля, а також, щоб була передбачена можливість автоматичної модифікації даних в одному файлі при зміні інформації в іншому файлі з метою забезпечення їх узгодженості.

Однак забезпечення узгодженості є одним з основних завдань, що покладається на систему управління базами даних, тобто розглянута вище інформаційна система має ознаки бази даних. Крім того, якщо інформаційна система підтримує узгоджене зберігання інформації у декількох файлах, то вона підтримує й базу даних. Таким чином, вимога підтримання узгодженості даних у декількох файлах не дозволяє обмежитися бібліотекою функцій: такій системі слід мати деякі власні дані (метадані), а також підтвердження цілісності даних.

Крім того, в інформаційній системі, яку наведено вище, незручно реалізовувати такі запити як "список всіх міст однієї країни". Доцільно формулювати такий запит зрозумілою користувачам мовою, яку називають мовою запитів до баз даних. При цьому метадані будуть містити інформацію про те, що поле **name** є ключовим для файла **town**, а **cntry_name** – для файла **country**, і система сама скористається цим. Якщо ж виникне потреба в отриманні списку міст, що знаходяться на території однієї країни, то досить подати системі запит на поле **name**, і

система сама виконає необхідний повний перегляд файла **town**, оскільки поле **country** не є ключовим.

Необхідно також зазначити, що інформаційна система, яку основано на використанні бібліотек розширених методів доступу до файлів, не забезпечує коректності стану баз даних у разі несподіваної відмови техніки, наприклад, аварійного виключення живлення. Вирішення завдання забезпечення коректності стану бази даних беруть на себе тільки системи управління базами даних.

Було зазначено, що файлові системи мають обмежені можливості щодо синхронізації паралельного доступу до даних. Отже, інформаційна система на базі файлової системи не може забезпечити користувачам паралельну роботу з базою даних. Якщо спиратися тільки на використання файлів, то для забезпечення коректної модифікації будь-якого з двох файлів одним користувачем, доступ інших користувачів до цього файла буде блоковано. Таким чином, при виконанні запиту одним користувачем отримання інформації іншим буде істотно затримано. Розвинені СУБД забезпечують набагато тоншу синхронізацію паралельного доступу до даних.

Таким чином, проведений аналіз свідчить, що створення інформаційних систем з використанням файлів не дозволяє повною мірою забезпечити ефективну роботу з даними. СУБД дозволяють вирішити велику кількість проблем, які важко або взагалі неможливо вирішити при використанні файлових систем. При цьому існують додатки, для яких є цілком достатньо файлів. Користувач вибирає системи, які йому слід використовувати для вирішення певних завдань щодо роботи з даними.

6.2 Основні функції СУБД

Можливості файлових систем значно обмежені для побудови навіть простих інформаційних систем. Система управління базами даних має створювати логічно узгоджений набір файлів, підтримувати мови маніпулювання даними, відновлювати інформацію після різного роду збоїв, забезпечувати паралельну роботу декількох користувачів. Якщо будь-яка система управління має ці властивості, то вона є системою управління базами даних.

Коротко розглянемо суть основних функцій СУБД, до яких належать:

- безпосереднє управління даними у зовнішній пам'яті;
- управління буферами оперативної пам'яті;
- управління транзакціями;
- журналізація;
- підтримання мов баз даних.

Функція безпосереднього управління даними у зовнішній пам'яті забезпечує створення необхідних структур зовнішньої пам'яті, які призначено для зберігання даних, що безпосередньо входять до бази

даних (БД), а також застосування цих структур у службових цілях, наприклад, для прискорення доступу до даних у деяких випадках, як правило, з використанням індексів. Деякі СУБД активно використовують можливості існуючих файлових систем, а деякі працюють на рівні пристроїв зовнішньої пам'яті. При цьому користувач не знає, використано в розвинених СУБД файлову систему або ні та як організовано файли. СУБД підтримує власну систему іменування об'єктів БД незалежно від користувача.

Функція управління буферами оперативної пам'яті забезпечує роботу з базами даних, що мають великі розміри, тобто їх розміри значно більше доступних обсягів оперативної пам'яті. Під час роботи з БД здійснюється звернення до елементів даних, тобто виконується обмін інформацією із зовнішньою пам'яттю. У зв’язку з цим швидкодія всієї системи залежить від швидкості пристрою зовнішньої пам'яті. В сучасних умовах найпоширенішим способом прискорення швидкодії є буферизація даних в оперативній пам'яті. Однак, навіть якщо операційна система проводить загальносистемну буферизацію, цього недостатньо для досягнення цілей СУБД, яка має більші можливості з буферизації частин БД. Тому у розвинених СУБД підтримується власний набір буферів оперативної пам'яті з власною організацією заміни буферів.

Найбільш важливою функцією СУБД є **функція управління транзакціями.** Транзакція – це послідовність операцій з БД, яка розглядається СУБД як єдине ціле. Транзакція або успішно виконується, і СУБД фіксує зміни БД у зовнішній пам'яті комп’ютера, проведені цією транзакцією (**COMMIT**), або жодна з цих змін ніяк не впливає на стан БД (**ROLLBACK**). Поняття транзакції є необхідним для підтримання логічної цілісності БД.

Властивість транзакції полягає в тому, що вона починається при цілісному стані БД і залишає цей стан цілісним після завершення. Поняття транзакції використовують як одиницю активності користувача відносно БД. При відповідному управлінні транзакціями, що паралельно виконуються, користувач може вважати себе єдиним користувачем СУБД.

Управління транзакціями в багатокористувальній СУБД здійснюється за допомогою їх серіалізацій за серіальним планом.

Під серіалізацією транзакцій, що виконуються паралельно, розуміють такий порядок планування їх роботи, при якому сумарний ефект поєднання транзакцій є еквівалентним ефекту їх деякого послідовного виконання.

Серіальний план виконання транзакцій – це такий план, який приводить до їх серіалізаціїї. Якщо вдається забезпечити дійсно серіальне виконання транзакцій, то для кожного користувача, за ініціативою якого утворено транзакцію, присутність інших транзакцій буде непомітною.

Функція журналізації спрямована на забезпечення надійності зберігання даних у зовнішній пам'яті. При цьому під надійністю зберігання

розуміють можливість СУБД відновити останній узгоджений стан БД після будь-якого апаратного або програмного збою.

Як правило, розрізняють два можливі види апаратних збоїв:

- м'які збої, які характеризуються раптовою зупинкою роботи комп'ютера, наприклад, аварійне вимикання живлення;
- жорсткі збої, що характеризуються втратою інформації на носіях зовнішньої пам'яті.

Прикладами програмних збоїв можуть бути:

- аварійне завершення роботи СУБД через помилку в програмі або в результаті деякого апаратного збою. Таку ситуацію можна розглядати як особливий вид м'якого апаратного збою;
- аварійне завершення призначеної для користувача програми, в результаті чого деяка транзакція залишається незавершеною. Під час виникнення цієї ситуації потрібно ліквідовувати наслідки тільки однієї транзакції.

Надійність зберігання даних у БД потребує застосування надмірних даних, причому та частина даних, яка використовується для відновлення, має зберігатися особливо надійно. Найбільш поширеним методом забезпечення такої надмірної інформації є ведення журналу змін БД.

Журнал – це особлива частина БД, яка є недоступною користувачам СУБД, яку зберігають з особливою ретельністю і в яку надходять записи про всі зміни основної частини БД. У деяких випадках зберігаються дві копії журналу, що розташовуються на різних фізичних дисках.

У всіх видах журналів реалізовується стратегія "попереджуючого" запису у журнал, протокол Write Ahead Log – WAL. Тобто запис про зміну будь-якого об'єкта БД має потрапити у зовнішню пам'ять журналу раніше, ніж змінений об'єкт потрапить у зовнішню пам'ять основної частини БД. Якщо в СУБД коректно дотримується протокол WAL, то за допомогою журналу можна вирішити всі проблеми відновлення БД після будь-якого збою.

Функція підтримання мов бази даних забезпечує створення схеми БД і маніпулювання даними в базах даних. Перші СУБД підтримували декілька спеціалізованих мов, які вирізнялися своїми функціями. Найчастіше використовували мову визначення схеми БД (SDL – Schema Definition Language) та мову маніпулювання даними (DML – Data Manipulation Language).

Мову SDL було призначено для визначення логічної структури БД, тобто тієї структури БД у вигляді якої її подають. Мова DML містить набір операторів маніпулювання даними, тобто операторів, що дозволяють заносити дані у БД, видаляти, модифікувати або вибирати існуючі дані.

У сучасних СУБД підтримується єдина інтегрована мова, що містить всі необхідні засоби для роботи з БД. Найбільш поширеною стандартною мовою реляційних СУБД є мова SQL (Structured Query Language).

Мова SQL поєднує функції мов SDL і DML, тобто дозволяє визначати схему реляційної БД і маніпулювати даними. При цьому іменування об'єктів БД підтримується на мовному рівні. Отже, компілятор мови SQL проводить перетворення імен об'єктів в їх внутрішні ідентифікатори на основі спеціальних підтримувальних службових таблиць-каталогів, а ядро СУБД не працює з іменами таблиць та їх стовпцями. У спеціальних таблицях-каталогах розміщено спеціалізовані засоби визначення обмежень цілісності БД, які підтримуються мовою SQL, тобто при компіляції операторів модифікації БД компілятор SQL на основі наявних у БД обмежень цілісності генерує відповідний програмний код. Крім того, спеціальні оператори мови SQL дозволяють здійснювати запити до реляційної БД, результатом яких є таблиці, що зберігаються у БД, з іменованими стовпцями. Нарешті авторизація доступу до об'єктів БД проводиться також на основі спеціального набору операторів SQL. Отже, повноваження користувачів описуються у спеціальних таблицях-каталогах, і контроль повноважень підтримується на мовному рівні.

Запитання для самоперевірки

1. Що являє собою файлова система ?
2. Як за допомогою файлової системи створити інформаційну систему ?
3. Які основні вимоги висуваються до системи управління БД ?
4. Які основні функції є властивими сучасним СУБД ?
5. Що таке транзакція ?
6. За допомогою чого підтримується надійність зберігання даних у БД?
7. Які основні функції виконує мова бази даних SQL ?

Лекція № 7

СТРУКТУРА СУЧАСНОЇ СУБД

Навчальні цілі:
- вивчити типову структуру сучасної СУБД;
- розглянути ранні підходи до створення структури баз даних.

Навчальні питання:
1. Ранні підходи до створення структури баз даних.
2. Типова структура сучасної СУБД.

7.1 Ранні підходи до створення структури баз даних

З метою більш глибокого розуміння принципів побудови сучасних БД коротко розглянемо ранні підходи до організації дореляційних СУБД. Крім того, організація сучасних СУБД базується на використанні методів побудови ранніх систем, які історично передували реляційним БД. Розуміння історичного розвитку СУБД дозволяє прогнозувати шляхи розвитку сучасних СУБД, які називають постреляційними СУБД.

Найбільш відомими серед ранніх СУБД є системи трьох типів:
- СУБД, основані на інвертованих списках;
- ієрархічні СУБД;
- мережні СУБД.

Існує декілька загальних характеристик ранніх систем.

З появою комп'ютерів і виникненням потреб у створенні інформаційних систем саме такі СУБД почали широко використовувати. За весь час їх застосування було накопичено великий обсяг баз даних, що управляються такими системами. Тому в наш час виникає потреба у забезпеченні сумісності існуючих архівів ранніх баз даних із сучасними СУБД.

Усі ранні системи не ґрунтувалися на абстрактних моделях. Абстрактні подання даних з'явилися пізніше на основі аналізу та виявлення загальних ознак у різних конкретних системах.

Крім того, у ранніх СУБД доступ до БД проводився на рівні записів. Користувачі цих систем здійснювали явну навігацію у БД, використовуючи мови програмування, що були розширені функціями СУБД. Інтерактивний доступ до БД підтримувався тільки шляхом створення відповідних прикладних програм з власним інтерфейсом. Отже, структура ранніх СУБД є дуже близькою до структури файлових систем. Така побудова систем привела користувачів до необхідності самостійно оптимізувати доступ до БД без будь-якого підтримання системи.

Раніше найбільш відомими серед СУБД були **системи, основані на інвертованих списках.** Прикладом таких СУБД є система Datacom/DB

компанії Applied Data Research, Inc. (ADR). Коротко розглянемо основні особливості таких систем.

Основою організації доступу до даних у всіх сучасних реляційних СУБД є принципи, які було закладено у системах на основі інвертованих списків. При цьому в реляційних СУБД користувачі не мають безпосереднього доступу до інвертованих списків (індексів), а в ранніх системах, що розглядаємо, така можливість була. Інтерфейси реляційних СУБД також є дуже близькими до інтерфейсів систем, основаних на інвертованих списках.

Структурно бази даних на основі інвертованих списків побудовано аналогічно реляційним БД. Відмінність полягає у тому, що в ранній СУБД таблиці, що зберігаються, а також шляхи доступу до них видно користувачам. При цьому рядки таблиць упорядковано системою у деякій послідовності, така впорядкованість рядків всіх таблиць може використовуватись і для всієї БД. У таких системах для кожної таблиці можна визначити довільне число ключів пошуку, для яких одержують індекси, що автоматично підтримуються системою. У той же час користувачі можуть явно бачити ці індекси.

Під час маніпулювання даними система, основана на інвертованих списках, підтримує два класи операторів.

Перший клас складають оператори, що встановлюють адресу запису. До цих операторів належать: прямі пошукові оператори (наприклад, знайти перший запис у таблиці за деяким шляхом доступу); оператори, що знаходять запис у термінах відносної позиції від попереднього запису за деяким шляхом доступу.

До другого класу належать оператори маніпулювання записами з адресами. Такі оператори забезпечують: пошук першого запису по порядку в таблиці або із заданим значенням ключа пошуку; пошук наступного запису після запису із заданою адресою; пошук наступного запису із заданим значенням ключа в таблиці; повернення адреси запису; вибір, відновлення, видалення запису за указаною адресою та ін.

У ранніх СУБД не існувало загальних правил визначення цілісності БД. У деяких системах підтримувалися обмеження унікальності значень деяких полів, але в основному все покладалося на прикладну програму.

Одним з ранніх представників СУБД є також **ієрархічні системи.** Прикладом таких СУБД є система Information Management System (IMS) фірми IBM. Коротко розглянемо основні особливості таких систем.

В основі побудови ієрархічних БД лежить ієрархічна модель даних, в якій є один основний (головний) об’єкт і декілька підпорядкованих об'єктів, що знаходяться на різних рівнях ієрархії. Така модель БД подається у вигляді зв'язного графа типу дерева, вершини якого розташовані на різних ієрархічних рівнях. При цьому одна з вершин на найвищому рівні називається коренем, а інші вершини з'єднані тільки з однією вершиною, яка розміщена на більш високому рівні.

Ієрархічна БД складається з упорядкованого набору декількох екземплярів одного типу дерева, а тип дерева – з одного кореневого типу запису і впорядкованого набору з нуля або більше типів піддерев. Структуру ієрархічної бази даних показано на рисунку 7.1.

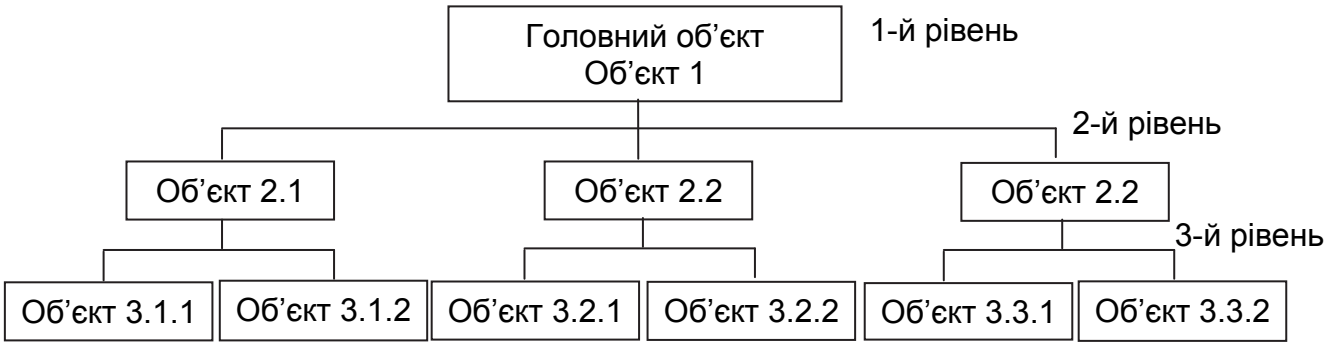


Рисунок 7.1 – Структура ієрархічної бази даних

Із рисунка 7.1 видно, що об’єкт 1 є предком об’єктів 2.1, 2.2 та 2.3, а об’єкти 2.1, 2.2 та 2.3 – нащадки об’єкта 1. Об’єкт 2.1 є предком об’єктів 3.1.1 та 3.1.2, а об’єкти 3.1.1 та 3.1.2 є нащадками об’єкта 2.1 і т.д. Між типами записів підтримуються зв'язки. Всі нащадки, що мають загального предка називаються близнюками. Так об’єкти 2.1, 2.2 та 2.3 є близнюками, також близнюками є об’єкти 3.1.1 та 3.1.2. Для БД визначається повна послідовність обходу – зверху вниз, зліва направо.

База даних з такою схемою може бути використана для подання, наприклад, штату будь-якої фірми. Головним об’єктом є назва та загальна кількість співробітників фірми. Об’єкти 2-го рівня можуть описувати назви та кількість співробітників філіалів фірми у різних містах країни, а об’єкти 3-го рівня – назви та кількість співробітників у структурних підрозділах філіалів фірми.

Під час маніпулювання даними в ієрархічній системі можуть застосовуватися типові оператори, наприклад:

- знайти вказане дерево БД;
- перейти від одного дерева до іншого;
- перейти від одного запису до іншого всередині дерева;
- ввести новий запис у зазначену позицію;
- видалити поточний запис та ін.

В ієрархічних системах автоматично підтримується цілісність посилань між предками та нащадками. При цьому реалізується основне правило, що ніякий нащадок не може існувати без свого предка (батька). Однак така цілісність записів, що не входять в одну ієрархію, не підтримується.

Більш розвиненим підходом до побудови ранніх СУБД є застосування мереж для створення систем управління, тобто так званих

мережних систем. Прикладом таких СУБД є система Integrated Database Management System (IDMS) компанії Cullinet Software, Inc.

Мережний підхід до організації даних є розширенням ієрархічного підходу. В ієрархічних системах запису-нащадку слід мати тільки одного предка, в той же час у мережних системах нащадок може мати будь-яку кількість предків. У мережній моделі даних будь-який об'єкт може бути одночасно і головним, і підпорядкованим. Кожний об'єкт може брати участь в утворенні будь-якої кількості взаємозв'язків з іншими об'єктами. Структуру мережної бази даних показано на рисунку 7.2.

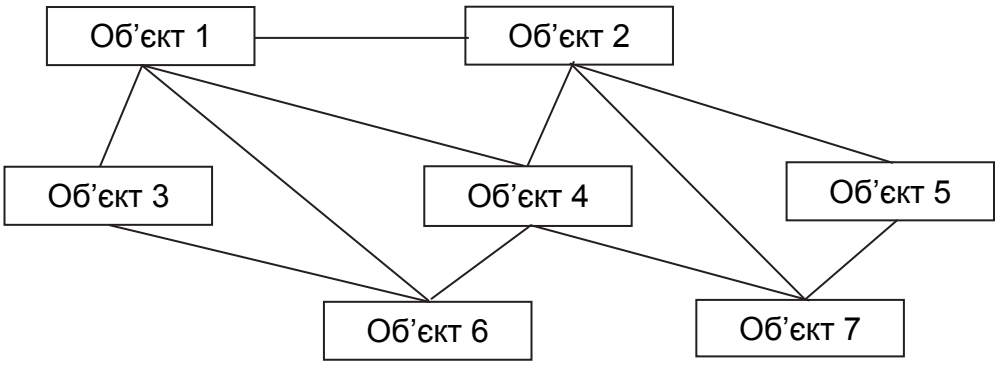


Рисунок 7.2 – Структура мережної бази даних

Таким чином, мережна БД складається з набору записів і набору зв'язків між цими записами. Тип зв'язку визначається для двох типів записів: предка та нащадка. Тип зв'язку складається з одного типу запису предка та впорядкованого набору типів записів нащадка.

Під час організації маніпулювання даними у мережній системі можуть застосовуватися типові оператори, наприклад:

- знайти конкретний запис у наборі однотипних записів;
- перейти від предка до першого нащадка за деяким зв'язком;
- перейти до наступного нащадка за деяким зв'язком;
- перейти від нащадка до предка за деяким зв'язком;
- створити новий запис;
- знищити запис;
- модифікувати запис та ін.

У мережних системах, як правило, підтримання цілісності не потрібне, але іноді є необхідним підтримання цілісності за посиланнями, як це робиться в ієрархічних системах.

Розглянемо загальні переваги та недоліки ранніх СУБД.

До переваг належать:

- розвинені на нижньому рівні засоби управління даними у зовнішній пам'яті, що спрощує їх аналіз і модифікацію;
- можливість побудови ефективних прикладних систем у ручному режимі;

- можливість економії пам'яті в результаті розділення підоб'єктів, особливо у мережних системах.

До недоліків належать:

- складність користування ранніми СУБД;
- необхідне знання фізичної організації таких систем;
- залежність прикладних систем, що створюють користувачі, від фізичної організації СУБД;
- ускладнена деталями побудова доступу до БД.

7.2 Типова організація сучасної СУБД

Організація типової СУБД і склад її компонентів відповідає основним функціям СУБД, які були розглянуті вище. Будь-яка СУБД складається з декількох програмних компонентів, кожен з яких призначено для виконання специфічної операції.

У сучасній СУБД виділяють:

- ядро (часто його називають Data Base Engine);
- компілятор мови БД (як правило, мови SQL);
- підсистему підтримання часу виконання;
- набір утиліт.

У деяких системах ці компоненти виділяються явно, в інших – ні, але логічно таке розподілення можна провести у всіх СУБД.

Ядро СУБД відповідає за управління даними у зовнішній пам'яті, буферами оперативної пам'яті і транзакціями, а також за журналізацію, тобто у ядрі виконуються перші чотири функції СУБД, які були розглянуті вище. Відповідно до цього виділяють чотири компоненти ядра: менеджер даних; менеджер буферів; менеджер транзакцій та менеджер журналу. Функції цих компонент є взаємозв'язаними, і для забезпечення коректної роботи СУБД вони мають взаємодіяти за допомогою ретельно продуманих та перевірених протоколів.

Ядро СУБД має свій власний інтерфейс, який безпосередньо є не доступним користувачам, але який використовують у програмах, що створюються компілятором мови SQL та утилітами БД. Ядро СУБД є основною резидентною частиною СУБД. Під час роботи у мережі з використанням архітектури "клієнт – сервер" ядро є основною складовою серверної частини системи.

Для компіляції операторів мови БД у деяку програму, яка може виконуватися, використовують **компілятор мови БД**. Необхідно зазначити, що мови, які застосовують у сучасних СУБД (найчастіше це мова SQL), є мовами не процедурними. Отже, під час виконання завдання оператор такої мови призначає деяку дію із БД, але в операторі лише описуються умови здійснення заданої дії, а він сам не є процедурою. У зв’язку з цим компілятор має вирішити, як реалізувати оператор мови перш ніж виконати програму. Результатом компіляції є програма, яку може

бути здійснено. У деяких системах цю програму можна подавати в машинних кодах, а у більшості СУБД – у внутрішньому машинно-незалежному коді.

Якщо в СУБД застосовується компілятор, що створює програму у внутрішньому машинно-незалежному коді, то реальне виконання оператора проводиться із залученням **підсистеми підтримання часу виконання**. Така підсистема по суті підтримує цю внутрішню мову СУБД, тобто є інтерпретатором цієї внутрішньої мови.

У сучасних СУБД існують процедури, які дуже невигідно виконувати з використанням мови БД. До таких процедур належать, наприклад, процедури завантаження та вивантаження БД, збору статистичних даних про БД, глобальної перевірки цілісності БД та ін. Для виконання таких процедур, як правило, створюють окремі **утиліти БД**. Утиліти програмують з використанням інтерфейсу ядра СУБД, а іноді навіть із проникненням всередину ядра.

Запитання для самоперевірки

1. У чому полягають загальні характеристики ранніх СУБД ?
2. Що є спільного і у чому різниця між СУБД на базі інвертованих списків та реляційних СУБД ?
3. У чому полягає суть ієрархічної моделі даних ?
4. У чому є різниця між ієрархічною та мережною БД ?
5. Які існують переваги та недоліки ранніх СУБД ?
6. Які функції виконує ядро СУБД ?
7. Для чого створюють окремі утиліти БД ?

Лекція № 8

ОСНОВНІ ПОНЯТТЯ РЕЛЯЦІЙНОЇ БАЗИ ДАНИХ

Навчальні цілі:
- вивчити базові поняття реляційних баз даних;
- розглянути фундаментальні властивості відношень.

Навчальні питання:
1. Базові поняття реляційних баз даних.
2. Фундаментальні властивості відношень.

8.1 Базові поняття реляційних баз даних

У наш час найбільш поширеним підходом до побудови баз даних є реляційний підхід. Перші прототипи реляційних СУБД з'явилися у 70-х роках минулого століття, але реалізація перших їх представників була не дуже ефективною. Поступове накопичення методів та алгоритмів організації реляційних баз даних та управління ними привели до того, що у середині 80-х років реляційні системи практично витіснили зі світового ринку ранні СУБД.

До основних переваг реляційного підходу належать:

- наявність невеликого набору абстракцій, які дозволяють порівняно просто моделювати велику частину поширених наочних областей і допускають точні формальні визначення, при цьому вони залишаються інтуїтивно зрозумілими;
- наявність простого, але потужного математичного апарата, що спирається на теорію множин і математичну логіку, які забезпечують теоретичний базис реляційного підходу до організації баз даних;
- можливість ненавігаційного маніпулювання даними без необхідності знання конкретної фізичної організації баз даних у зовнішній пам'яті.

До основних недоліків реляційної СУБД належать:

- деяка обмеженість при їх використанні у так званих нетрадиційних областях (найбільш поширеними прикладами є системи автоматизації проектування), в яких потрібні дуже складні структури даних;
- неможливість адекватного відображення семантики наочної області, тобто можливості подання знань про семантичну специфіку наочної області у реляційних системах є дуже обмеженими.

До основних понять реляційних баз даних належать: **тип даних; домен; схема відношення; кортеж; атрибут; відношення; первинний ключ; зовнішній ключ**.

Коротко розглянемо суть цих понять на прикладі відношення **town**, яке наведено у таблиці 8.1.

Таблиця 8.1 – Приклад відношення **town**

	Типи даних							
	Рядок	Рядок	Число	Число	Число	Логічна	Число	
	Домени							
	Місто	Країна	Населення	Довгота	Широта	Є столицею	Населення області	
	Атрибути							
	Первинний ключ	Зовнішній ключ						
ВІДНОШЕННЯ	Name	Country	Population	Dol	Shir	Capital	Pop_urb	**КОРТЕЖІ**
	Kharkov	Ukraine	1 940 000	36,158	50,094	N	2384,6	
	Donets'k	Ukraine	2 200 000	37,811	48,069	N	4568	
	Warsaw	Poland	2 323 000	21,130	52,225	Y	3703,5	
	Moskva	Russia	13 100 000	37,700	55,750	Y		
	Volgograd	Russia	1 360 000	44,472	48,655	N		
	Vienna	Austria	1 875 000	16,281	48,175	Y		
	Sofia	Bulgaria	1 205 000	23,315	42,686	Y		
	Kazan'	Russia	1 140 000	49,162	55,583	N		
	Kiev	Ukraine	2 900 000	30,492	50,453	Y		
	Tallinn	Estonia	482 000	24,754	59,313	Y		

Тип даних. Поняття тип даних є повністю адекватним поняттю типу даних у мовах програмування. Як правило, у сучасних реляційних БД допускається зберігання символьних і числових даних, бітових рядків, спеціалізованих числових даних, а також спеціальних «темпоральних» даних (дати, часу, часового інтервалу). Активно розвивається також підхід до розширення можливостей реляційних БД абстрактними типами даних.

Домен. Поняття домену є більш специфічним для реляційних баз даних. Якщо розглядати деякі мови програмування, можна знайти деяку аналогію домену з підтипами даних. Можна стверджувати, що домен визначається заданням деякого базового типу даних, до якого належать елементи домену, а також довільного логічного виразу, якому відповідає елемент цього типу даних. Якщо обчислення цього логічного виразу приводить до результату «істина», то елемент даних є елементом домену.

Поняття «домен» означає допустиму потенційну безліч значень даного типу. Необхідно зазначити, що дані можна порівнювати між собою тільки в тому випадку, якщо вони належать до одного домену.

Схема відношення. Схема відношення являє собою іменовану безліч пар {ім'я атрибута, ім'я домену}. Якщо поняття домену у БД не підтримується, схема відношення – це іменована безліч пар {ім'я атрибута, ім'я типу}. Під час розглядання поняття «схема відношення» вводять поняття «ступінь», або «арність» схеми відношення. Відповідно до теорії множин «арність» схеми відношення – це потужність цієї множини. Наприклад, ступінь відношення **town** дорівнює семи (кількості атрибутів), тобто воно є 7-арним. Якщо всі атрибути одного відношення визначені на

різних доменах, доцільно та виправдано використовувати для названня атрибутів імена відповідних доменів.

Коли розглядають структуру БД, вводять поняття схеми БД, яка є набором іменованих схем відношень.

Кортеж являє собою безліч (велику кількість) пар {ім'я атрибута, значення} і містить одне входження кожного імені атрибута, що належить схемі відношення. Значення кортежу є допустимим значенням домену даного атрибута. «Ступінь», або «арність» кортежу, тобто кількість елементів, з яких він складається, збігається з «арністю» відповідної схеми відношення. Іншими словами, кортеж – це набір іменованих значень заданого типу.

Атрибут відповідає властивостям об'єктів, відомості про які зберігаються у таблиці реляційної БД. У деяких СУБД атрибути називають полями.

Відношення являє собою велику кількість (безліч) кортежів, які відповідають одній схемі відношення. Іноді цю схему відношення називають заголовком відношення, а саме відношення як набір кортежів – тілом відношення.

Найбільш часто відношення подають як таблицю, заголовком якої є схема відношення, рядками – кортежі відношення, а стовпці цієї таблиці – імена атрибутів. Іноді говорять «стовпець таблиці», маючи на увазі атрибут відношення.

Таким чином, **реляційна база даних** являє собою набір відношень, імена яких збігаються з іменами схем відношень у схемі БД.

Наведені вище поняття в теорії реляційних БД визначаються абсолютно формально й точно.

8.2 Фундаментальні властивості відношень

Основним поняттям реляційної БД є поняття відношення. Розглянемо деякі важливі властивості відношень.

Відсутність кортежів-дублікатів. Властивість, що відношення не містять кортежів-дублікатів, пов’язана з визначенням відношення як безлічі (великої кількості) кортежів. Як відомо з класичної теорії множин, кожна множина складається з різних елементів.

Ця властивість дозволяє надавати кожному відношенню так званий первинний ключ. **Первинний ключ** являє собою набір атрибутів, значення яких однозначно визначають кортеж відношення. Для кожного відношення повний набір його атрибутів має цю властивість. Однак при формальному визначенні первинного ключа необхідно забезпечувати його мінімальність. Мінімальність передбачає те, що у набір атрибутів первинного ключа не мають входити такі атрибути, які можна відкинути без утрати однозначності визначення кортежу.

Необхідно зазначити, що у деяких реалізаціях реляційної СУБД допускається порушення властивості відсутності кортежів-дублікатів для проміжних відношень, які породжуються неявно при виконанні запитів. При цьому такі відношення не є множинами, а являють собою мультимножину. Порушення цієї властивості іноді може привести до серйозних проблем.

Відсутність упорядкованості кортежів. Ця властивість також пов’язана з визначенням відношення як безлічі (великої кількості) кортежів. Відсутність вимоги до підтримання порядку на безлічі кортежів відношення надає додаткової гнучкості СУБД під час зберігання баз даних у зовнішній пам'яті, а також під час виконання запитів до бази даних. Відсутність упорядкованості кортежів не означає, що це буде заважати сортуванню результуючої таблиці відповідно до значень деяких стовпців під час формулювання такого запиту до БД. Результат цього запиту буде не відношенням, а поданням деякого впорядкованого списку кортежів.

Відсутність упорядкованості атрибутів. Ця властивість пов’язана з визначенням схеми відношення як безлічі пар {ім'я атрибута, ім'я домену}. При цьому атрибути відношень будуть невпорядкованими. Для посилання на значення атрибута в кортежі відношення завжди використовується ім'я атрибута. Властивість відсутності впорядкованості атрибутів теоретично дозволяє, наприклад, модифікувати схеми існуючих відношень не тільки шляхом додавання нових атрибутів, але й шляхом видалення існуючих атрибутів. Необхідно зазначити, що у більшості існуючих СУБД така можливість не допускається. Часто як неявний порядок атрибутів використовується їх порядок у лінійній формі визначення схеми відношення, хоча впорядкованість набору атрибутів відношення явно не потрібна.

Атомарність значень атрибутів. Значення всіх атрибутів у відношенні є атомарними. Ця властивість пов’язана з визначенням домену як потенційної безлічі значень простого типу даних, тобто серед значень домену не може міститися безліч значень (відношень). У реляційних БД даних допускаються тільки нормалізовані відношення або відношення, подані у першій нормальній формі. Приклад ненормалізованого відношення наведено у таблиці 8.2.

Із таблиці 8.2 видно, що це відношення не є атомарним, а є бінарним, значенням атрибута **country** відповідають відношення. Початкове нормалізоване відношення **town** наведено у таблиці 8.3.

Таким чином, нормалізовані відношення складають основу класичного реляційного підходу до організації баз даних. Вони мають деякі обмеження, наприклад, не всю інформацію зручно подавати у вигляді плоских таблиць, але вони істотно спрощують маніпулювання даними.

Таблиця 8.2 – Приклад ненормалізованого відношення **town**

Country	Name	Population	Dol	Shir	Capital	Pop_urb
Ukraine	Kharkov	1 940 000	36,158	50,094	N	2384,6
	Donets'k	2 200 000	37,811	48,069	N	4568
	Kiev	2 900 000	30,492	50,453	Y	3703,5
Poland	Warsaw	2 323 000	21,130	52,225	Y	
Russia	Moskva	13 100 000	37,700	55,750	Y	
	Volgograd	1 360 000	44,472	48,655	N	
	Kazan'	1 140 000	49,162	55,583	N	
Austria	Vienna	1 875 000	16,281	48,175	Y	
Bulgaria	Sofiya	1 205 000	23,315	42,686	Y	
Estonia	Tallinn	482 000	24,754	59,313	Y	

Таблиця 8.3 – Початкове нормалізоване відношення **town**

Name	Country	Population	Dol	Shir	Capital	Pop_urb
Kharkov	Ukraine	1 940 000	36,158	50,094	N	2384,6
Donets'k	Ukraine	2 200 000	37,811	48,069	N	4568
Warsaw	Poland	2 323 000	21,130	52,225	Y	
Moskva	Russia	13 100 000	37,700	55,750	Y	
Volgograd	Russia	1 360 000	44,472	48,655	N	
Vienna	Austria	1 875 000	16,281	48,175	Y	
Sofiya	Bulgaria	1 205 000	23,315	42,686	Y	
Kazan'	Russia	1 140 000	49,162	55,583	N	
Kiev	Ukraine	2 900 000	30,492	50,453	Y	3703,5
Estonia	Tallinn	482 000	24,754	59,313	Y	

Запитання для самоперевірки

1. У чому полягають переваги та недоліки реляційних СУБД ?
2. У чому полягають особливості поняття домену в реляційній БД ?
3. Як визначається «арність» схеми відношення ?
4. Як у відношенні визначаються кортежі й атрибути ?
5. Що таке мінімальність первинного ключа ?
6. Що дозволяє властивість відсутності впорядкованості кортежів ?
7. У чому полягає різниця між ненормалізованим і нормалізованим відношеннями ?

Лекція № 9

РЕЛЯЦІЙНА МОДЕЛЬ ДАНИХ

Навчальні цілі:
- вивчити структуру реляційної моделі даних;
- розглянути основні операції реляційної алгебри.

Навчальні питання:
1. Реляційна модель даних.
2. Основні операції реляційної алгебри.

9.1 Реляційна модель даних

Будь-яка модель даних описує деякий набір базових понять та ознак, який мають всі конкретні СУБД і керовані ними бази даних, якщо вони ґрунтуються на цій моделі. Наявність моделі даних дозволяє порівнювати конкретні реалізації, використовуючи одну спільну мову.

Найбільш поширене трактування реляційної моделі даних належить Кристоферу Дейту, який практично розвинув реляційний підхід засновника реляційних БД Эдгара Кодда. Згідно з трактуванням Дейта реляційна модель складається з трьох частин, що описують різні аспекти реляційного підходу.

Основними частинами реляційної моделі є:
- структурна;
- маніпуляційна;
- цілісна.

У **структурній частині** моделі визначається, що єдиною структурою даних, яка використовується у реляційних БД, є нормалізовані **n**-арні відношення. У структурній частині також наводяться поняття доменів, атрибутів, кортежів, схеми відношення і відношення. Отже, у структурній частині моделі розглядаються ті поняття й властивості реляційної БД, що були наведені у попередній лекції.

У **маніпуляційній частині** моделі визначаються два фундаментальні механізми маніпулювання реляційними БД – реляційна алгебра та реляційне числення. Перший механізм базується в основному на класичній теорії множин, а другий – на математичній логіці, зокрема класичному логічному апараті числення предикатів першого порядку та числення доменів.

Необхідно зазначити, що ці механізми мають одну важливу властивість: вони замкнуті щодо поняття відношення. Отже, вирази реляційної алгебри та формули реляційного числення визначаються для відношень реляційних БД і результатом обчислення також є відношення.

При цьому будь-який вираз або формула являють собою відношення, що дозволяє використовувати їх в інших виразах або формулах.

Реляційна алгебра або реляційне числення дозволяє складні запити до БД подати за допомогою одного виразу або однієї формули. Завдяки цьому ці механізми розміщено у реляційній моделі даних. Конкретна мова маніпулювання реляційними БД називається реляційно повною, якщо будь-який запит, що задається за допомогою одного виразу реляційної алгебри або однієї формули реляційного числення, може бути подано за допомогою одного оператора цієї мови.

Необхідно зазначити, що механізми реляційної алгебри та реляційного числення є еквівалентними, тобто для будь-якого допустимого виразу реляційної алгебри можна побудувати еквівалентну формулу реляційного числення і навпаки. Вони розрізняються тільки рівнем процедурності. Так запит, що задається на мові реляційної алгебри, може бути визначено на основі обчислення елементарних операцій алгебри з урахуванням їх старшинства та можливої наявності дужок. У свою чергу за допомогою формули реляційного числення тільки встановлюють умови, яким мають задовольняти кортежі результуючого відношення. Тому мови реляційного числення є більш непроцедурними, або декларативними.

Як правило, будь-яка мова БД, наприклад, мова SQL, ґрунтується на деякій сукупності конструкцій реляційної алгебри та реляційного числення.

У **цілісній частині** моделі визначаються дві базові вимоги цілісності, які мають підтримуватися у будь-якій реляційній СУБД, – вимога цілісності суті та вимога цілісності за посиланнями.

Вимога цілісності суті. Об'єкту або сутності реального миру в реляційних БД відповідають кортежі відношень. Вимога цілісності суті полягає у тому, що будь-який кортеж будь-якого відношення має бути відмінним від будь-якого іншого кортежу цього відношення, тобто будь-якому відношенню слід мати первинний ключ. Ця вимога автоматично задовольняється, якщо в системі не порушуються базові властивості відношень.

Іншими словами, вимогу цілісності суті можна сформулювати так: необхідно, щоб у будь-якого відношення існував первинний ключ, і ніяке значення первинного ключа у кортежах відношення не має містити невизначених значень (**NULL**).

Коротко розглянемо поняття **невизначене значення** (**NULL**). Як правило, будь-який кортеж, що вноситься у відношення, має містити всі характеристики модельованої ним сутності реального миру, які необхідно зберегти у базі даних. На практиці існують ситуації, коли на момент фіксації сутності у базі даних не всі її характеристики можуть бути відомими користувачеві. У цьому випадку використовують невизначені значення, які не належать ніякому типу даних і можуть існувати серед значень будь-якого атрибута, визначеного за допомогою будь-якого типу даних.

Таким чином, вимога цілісності суті означає, що первинний ключ має повністю ідентифікувати кожну сутність, а тому у складі будь-якого значення первинного ключа не допускається наявність невизначених значень.

Для дотримання цілісності суті достатньо гарантувати відсутність у будь-якому відношенні кортежів з одним і тим же значенням первинного ключа.

Вимога цілісності за посиланнями. Більш складна сутність реального миру подається у реляційній БД у вигляді декількох кортежів декількох нормалізованих відношень. При цьому зазначають, що відношення, в якому визначено **зовнішній ключ**, посилається на відповідне відношення, в якому такий же атрибут є первинним ключем.

Зовнішнім ключем називається атрибут (можливо, складений), значення якого однозначно характеризує сутність, подану кортежами деякого іншого відношення, тобто задають значення їх первинного ключа. Як правило, зовнішній ключ є складеним, тобто містить декілька атрибутів.

Вимога цілісності за посиланнями, або вимога зовнішнього ключа полягає в тому, що для кожного значення зовнішнього ключа, що визначається у відношенні, яке посилається, і у відношенні, на яке є посилання, має знайтися кортеж з таким же значенням первинного ключа, або значення зовнішнього ключа має бути невизначеним.

Коротко розглянемо, як забезпечується дотримання цілісності за посиланнями. Так, під час оновлення відношення (встановлення нових кортежів або модифікація значення зовнішнього ключа в існуючих кортежах), що посилається, достатньо стежити за тим, щоб не з'являлися некоректні значення зовнішнього ключа. Однак, як цього досягти, коли здійснюється видалення кортежу з відношення, на яке існує посилання ? При цьому існують три підходи.

Перший підхід полягає в тому, що забороняється проводити видалення кортежу, на який існують посилання. Спочатку потрібно або видалити кортежі, що посилаються, або відповідним чином змінити значення їх зовнішнього ключа.

При **другому підході** під час видалення кортежу, на який є посилання, у всіх кортежах, що посилаються, значення зовнішнього ключа автоматично стають невизначеними.

Третій підхід, який ще називають каскадне видалення, полягає в тому, що під час усунення кортежу з відношення, на яке ведеться посилання, із відношення, що посилається, автоматично видаляються всі кортежі, на які є посилання.

У розвинених реляційних СУБД зазвичай можна вибрати спосіб підтримання цілісності за посиланнями для кожної окремої ситуації визначення зовнішнього ключа. Звичайно, для ухвалення такого рішення необхідно аналізувати вимоги щодо конкретної прикладної області.

9.2 Основні операції реляційної алгебри

Основна ідея реляційної алгебри полягає у тому, що якщо відношення є множинами, то засоби маніпулювання відношеннями можуть базуватися на традиційних теоретико-множинних операціях, доповнених деякими спеціальними операціями, специфічними для баз даних. При цьому набір основних операцій алгебри складається з восьми операцій, які поділяють на два класи: теоретико-множинні та спеціальні реляційні.

До складу **теоретико-множинних операцій** входять операції:

- об'єднання відношень;
- перетину відношень;
- утворення різниці відношень;
- прямого добутку відношень.

Коротко розглянемо суть цих операцій.

Об'єднання відношень. При виконанні операції об'єднання двох відношень створюється відношення, що містить усі кортежі, що входять хоча б в одне з відношень-операндів.

Сенс операції об'єднання у реляційній алгебрі в цілому залишається теоретико-множинним. Але, якщо у теорії множин операція об'єднання є справедливою для будь-яких двох множин-операндів, то у реляційній алгебрі результатом операції об'єднання має бути відношення. Якщо допустити у реляційній алгебрі можливість теоретико-множинного об'єднання довільних двох відношень (з різними схемами), то результатом операції буде множина різнотипних кортежів, тобто не відношення. Якщо виходити з вимоги замкнутості реляційної алгебри щодо поняття відношення, то така операція об'єднання є неприпустимою.

З цієї ситуації витікає поняття **сумісності відношень за об'єднанням:** два відношення є сумісними за об'єднанням у тому і лише у тому випадку, коли мають однакові заголовки. Це означає, що у заголовках обох відношень має бути один і той же набір імен атрибутів, та однойменні атрибути слід визначати на одному й тому ж домені.

Перетин відношень. Операція перетину двох відношень створює відношення, що містить усі кортежі, що входять в обидва відношення-операнди.

Утворення різниці відношень. Відношення, що є різницею двох відношень, містить усі кортежі, які входять у відношення, що є першим операндом, при цьому жодний з них не входить у відношення, що є другим операндом.

Все, що було розглянуто вище стосовно операції об'єднання, є справедливим для операцій перетину та утворення різниці відношень.

Якщо два відношення є сумісними за об'єднанням, то при звичайному виконанні з ними операцій об'єднання, перетину та утворення різниці відношень результатом операції буде відношення з коректним заголовком, що збігається із заголовком кожного з відношень-операндів.

Прямий добуток відношень. При виконанні прямого добутку двох відношень створюється відношення, кортежі якого є конкатенацією (зчепленням або злиттям) кортежів першого і другого операндів.

Операція прямого добутку двох відношень має інші особливості. У теорії множин прямий добуток може бути отримано для будь-яких двох множин, та елементами результуючої множини є пари, складені з елементів першої і другої множини. Оскільки відношення є множинами, то і для будь-яких двох відношень можливе отримання прямого добутку. Однак результат не буде відношенням, тому що результатом будуть не кортежі, а пари кортежів.

У реляційній алгебрі використовують спеціалізовану форму операції прямого добутку двох відношень – **розширений прямий добуток відношень**. При застосуванні розширеного прямого добутку двох відношень елементом результуючого відношення є кортеж, що є конкатенацією (зчепленням або злиттям) одного кортежу першого відношення з одним кортежем другого відношення.

Необхідно зазначити, що під час виконання цієї операції необхідно коректно сформувати заголовок відношення-результату, тобто здійснити правильне іменування атрибутів результуючого відношення, якщо відношення-операнди мають однойменні атрибути. При цьому вводиться поняття **сумісності щодо утворення розширеного прямого добутку**. Два відношення є сумісними щодо утворення прямого добутку у тому і лише у тому випадку, якщо безліч імен атрибутів цих відношень не перетинаються. Будь-які два відношення можна зробити сумісними при застосуванні операції перейменування до одного з цих відношень.

Операція утворення прямого добутку на практиці застосовується не дуже часто. Це пов’язано з тим, що потужність результуючого відношення є достатньо великою навіть тоді, коли потужності операндів-відношень є припустимими. Також це пов’язано з тим, що результат операції не є більш інформативним, ніж результат утворення сукупності операндів-відношень. Однак на базі операції утворення розширеного прямого добутку визначається операція реляційної алгебри – з'єднання.

Необхідно зазначити, що всі чотири операції, що розглянуті вище, є асоціативними, тобто, якщо позначити через OP будь-яку з цих чотирьох операцій, то (A OP B) OP C = A OP (B OP C). У цьому виразі A, B і C є відношення, що мають властивості, які потрібні для коректного виконання відповідної операції. Всі операції, окрім утворення різниці, є комутативними, тобто A OP B = B OP A.

Спеціальні реляційні операції містять:

- обмеження відношення;
- проекцію відношення;
- з'єднання відношень;
- поділення відношень.

Коротко розглянемо суть цих операцій.

Обмеження відношення. Результатом обмеження відношення за деякою умовою є відношення, що містить кортежі відношення-операнда, які задовольняють цій умові. Операція обмеження потребує наявності двох операндів: відношення, яке обмежується, та простої умови обмеження. Іншими словами, операцію обмеження можна подати як одержання деякої "горизонтальної" вирізки з відношення-операнда.

Проста умова обмеження може мати вигляд ***a*** **comp-op** ***b***, де ***a*** і ***b*** – імена атрибутів відношення, які обмежуються, до яких може бути застосована операція порівняння **comp-op**. Умова обмеження може мати вигляд ***a*** **comp-op** ***const***, де ***a*** – ім'я атрибута відношення, який обмежується, а ***const*** – літерально задана константа.

У результаті виконання операції обмеження створюється відношення, заголовок якого збігається із заголовком відношення-операнда, а в тіло входять ті кортежі відношення-операнда, для яких значенням умови обмеження є **true** (істина). Із застосуванням цих визначень можна використовувати операції обмеження, в яких умовою обмеження є довільний булевий вираз, що складено з простих умов із введенням логічних зв'язок **AND**, **OR**, **NOT** і дужок.

Проекція відношення. При виконанні проекції відношення на заданий набір його атрибутів одержуємо відношення, кортежі якого створюються шляхом вибору відповідних значень з кортежів відношення-операнда.

Операція утворення проекції також потребує наявності двох операндів: відношення ***A***, яке проектується, та списку імен атрибутів, що входять у заголовок відношення ***A***. Іншими словами, при виконанні операції проекції виділяється "вертикальна" вирізка відношення-операнда з природним знищенням потенційно виникаючих кортежів-дублікатів. Результатом проекції відношення ***A*** за списком атрибутів $a_1, a_2, ..., a_n$ є відношення із заголовком, який визначається безліччю атрибутів $a_1, a_2, ..., a_n$, та з тілом, що складається з кортежів $<a_1:v_1, a_2:v_2, ..., a_n:v_n>$. Отже, у відношенні ***A*** є кортеж, атрибут a_1 якого має значення v_1, атрибут a_2 має значення v_2, ..., атрибут a_n – значення v_n.

З'єднання відношень. При з'єднанні двох відношень за деякою умовою утворюється результуюче відношення, кортежі якого є конкатенацією кортежів першого і другого відношень і задовольняють цій умові.

Операція з'єднання потребує наявності трьох операндів: двох відношень, що об'єднуються, та простої умови. Наприклад, якщо з'єднуються відношення ***A*** та ***B***, то так само, як і у разі операції обмеження, умова з'єднання **comp** має вигляд або ***a*** **comp-op** ***b***, або **a comp-op** ***const***, де ***a*** та ***b*** – імена атрибутів відношень ***A*** та ***B***, ***const*** – літерально задана константа, *a* **comp-op** – допустима операція порівняння.

Таким чином, результатом операції з'єднання є відношення, що отримують шляхом виконання операції обмеження за умови **comp** прямого добутку відношень ***A*** та ***B***.

Необхідно зазначити, що застосування операції з'єднання зменшує потужність результату проміжного прямого добутку відношень-операндів, але тільки у тому випадку, коли умова з'єднання має вигляд ***a* comp-op *b***, де ***a*** та ***b*** – імена атрибутів різних відношень-операндів. На практиці, як правило, використовують операції з'єднання, які ґрунтуються саме на цій умові з'єднання.

На практиці також часто застосовують окремий випадок операції з'єднання – еквіз'єднання, а також розширення операції еквіз'єднання – природне з'єднання. Операція з'єднання називається **операцією еквіз'єднання**, якщо умова з'єднання має вигляд ***a* = *b***, де ***a*** і ***b*** – атрибути різних операндів з'єднання. Для такого виду з'єднання існують ефективні алгоритми реалізації. Операцію **природного з'єднання** застосовують до пари відношень ***A*** та ***B***, що мають загальний атрибут (можливо складений), тобто атрибут з одним і тим же ім'ям та визначеним на одному і тому ж домені. Наприклад, якщо позначити ***ab*** як об'єднання заголовків відношень ***A*** та ***B***, то природне з'єднання ***A*** та ***B*** є спроектованим на ***ab*** результатом еквіз'єднання ***A*** та ***B***. Основний сенс операції природного з'єднання – це можливість відновлення складної сутності відношення, яка була декомпозована через потребу вимоги першої нормальної форми.

Поділення відношень. В операції реляційного поділення беруть участь два операнди – бінарне і унарне відношення. Результуюче відношення складається з одноатрибутних кортежів, що містять значення першого атрибута кортежів першого операнда, таких, що безліч значень другого атрибута (при фіксованому значенні першого атрибута) збігаються з безліччю значень другого операнда.

Операцію поділення відношень можна пояснити. Нехай задано два відношення: перше відношення ***A*** із заголовком {$a_1, a_2, ..., a_n, b_1, b_2, ..., b_m$} і друге відношення ***B*** із заголовком {$b_1, b_2, ..., b_m$}. Будемо вважати, що атрибут b_i відношення ***A*** і атрибут b_i відношення ***B*** не тільки мають одне й те ж ім'я, але й визначені на одному й тому ж домені. Безліч атрибутів {a_j} будемо називати складеним атрибутом ***a***, а безліч атрибутів {b_j} – складеним атрибутом ***b***. Після цього можна розглядати реляційне поділення бінарного відношення ***A*****(*a*, *b*)** на унарне відношення ***B*****(*b*)**.

Результатом поділення бінарного відношення ***A*** на унарне відношення **B** є унарне відношення ***C*****(*a*)**, що складається з кортежів ***v***. Отже, у відношенні ***A*** є кортежі <***v***, ***w***> такі, що безліч значень {***w***} містить безліч значень атрибуту ***b*** відношення ***B***.

Крім того, до складу реляційної алгебри входять операції перейменування атрибутів і присвоювання.

Коротко розглянемо суть цих операцій.

Операція перейменування атрибутів. Ця операція дає можливість коректно сформувати заголовок (схему) результуючого відношення. Операція перейменування створює відношення, тіло якого збігається з тілом операнда, але імена атрибутів змінено.

Операція присвоювання. Ця операція дозволяє зберегти результат обчислення реляційного виразу в існуючому відношенні БД.

Таким чином, результатом будь-якої реляційної операції (окрім операції присвоювання) є деяке відношення. Це дозволяє створювати реляційні вирази, в яких замість відношення-операнда деякої реляційної операції знаходиться вкладений реляційний вираз.

Запитання для самоперевірки

1. Які поняття визначаються у структурній частині реляційної моделі ?
2. На чому базується маніпулювання даними у реляційних БД ?
3. У чому полягає різниця між реляційною алгеброю та реляційним численням ?
4. У чому існує різниця між вимогами цілісності суті та цілісності за посиланнями ?
5. У чому полягають особливості застосування теоретико-множинних операцій реляційної алгебри ?
6. В яких випадках застосовується операція перейменування атрибутів ?
7. Які окремі випадки має операція з'єднання ?

Лекція № 10

ВНУТРІШНЯ ОРГАНІЗАЦІЯ РЕЛЯЦІЙНИХ СУБД

Навчальні цілі:
- розглянути структури зовнішньої пам’яті;
- вивчити методи організації індексів.

Навчальні питання:
1. Структури зовнішньої пам’яті.
2. Методи організації індексів.
3. Зберігання журнальної та службової інформації.

10.1 Структури зовнішньої пам'яті

Реляційні СУБД мають низку особливостей, що впливають на організацію зовнішньої пам'яті.

По-перше, система має два рівня: рівень безпосереднього управління даними у зовнішній пам'яті та мовний рівень. На першому рівні, як правило, здійснюється управління буферами оперативної пам'яті, управління транзакціями та журналізація змін БД. На другому рівні реалізується мова БД, наприклад, мова SQL. У результаті такої побудови підсистема нижнього рівня має підтримувати у зовнішній пам'яті набір базових структур, конкретна інтерпретація яких входить до числа функцій підсистеми верхнього рівня.

По-друге, система підтримує відношення-каталоги. При цьому інформація, яка пов'язана з іменуванням об'єктів бази даних та їх конкретними властивостями, підтримується підсистемою мовного рівня; щодо структур зовнішньої пам'яті відношення-каталог не відрізняється від звичайного відношення бази даних.

По-третє, система забезпечує регулярність структур даних. Це обумовлюється тим, що основним об'єктом реляційної моделі даних є плоска таблиця і виходячи з цього головний набір об'єктів зовнішньої пам'яті може мати дуже просту регулярну структуру. При цьому для ефективного виконання операторів мовного рівня у зовнішній пам'яті підтримуються додаткові управляючі структури – індекси. За допомогою цього механізму забезпечується можливість виконання операторів як з одним відношенням, наприклад, здійснення простої селекції або проекції, так і з декількома відношеннями, наприклад, здійснення з'єднання декількох відношень.

Крім того, система підтримує надмірність зберігання даних, яке спрямоване на виконання вимоги надійного зберігання баз даних. Для надійного зберігання даних у реляційних БД використовується журнал змін бази даних.

На основі розглянутих вище особливостей реляційної СУБД обумовлено різновиди об'єктів у зовнішній пам'яті бази даних. До об’єктів, що зберігаються у зовнішній пам'яті БД, належать:

- рядки відношень – основна частина бази даних, яка, як правило, може безпосередньо спостерігатися користувачами;
- управляючі структури – індекси, що створюються за ініціативою користувачів (адміністратора) або верхнього рівня системи виходячи з міркувань підвищення ефективності виконання запитів та які, як правило, автоматично підтримуються нижнім рівнем системи;
- журнальна інформація, яку спрямовано на забезпечення надійного зберіганні даних;
- службова інформація, яку використовують для вирішування внутрішніх завдань нижнього рівня системи, наприклад, інформація про вільну пам'ять.

Коротко розглянемо, як зберігаються у зовнішній пам’яті основні об’єкти БД.

Зберігання відношень. У наш час існують два принципові підходи до фізичного зберігання відношень. Найбільш поширеним є **покортежне зберігання відношень** (перший підхід) . Такий підхід забезпечує швидкий доступ до цілого кортежу, але при цьому у зовнішній пам'яті дублюються загальні значення різних кортежів одного відношення. Крім того, під час реалізації такого підходу потрібні зайві обміни з зовнішньою пам'яттю, якщо потрібно вибрати частину кортежу.

Типову структуру сторінки даних під час організації покортежного зберігання відношень показано на рисунку 10.1.

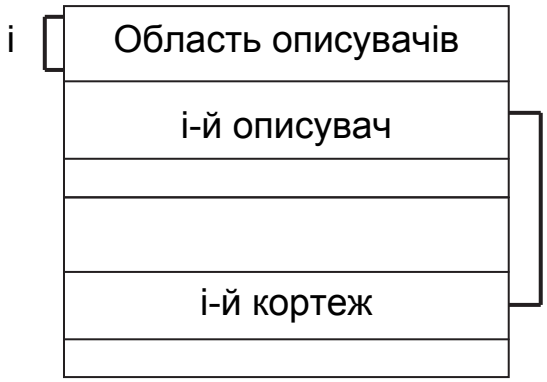


Рисунок 10.1 – Структура сторінки даних під час покортежного зберігання відношень

Структура покортежного зберігання відношень має певні характеристики:

- **кожен кортеж має унікальний ідентифікатор (tid)**, який не змінюється за весь час існування кортежу;
- **кожен кортеж, як правило, зберігається цілком на одній сторінці.** Однак розмір сторінки є обмеженим, тому максимальна довжина

кортежу будь-якого відношення також обмежена розмірами сторінки. При цьому у разі необхідності зберігати “довгі” дані, які не поміщаються на одній сторінці, використовують декілька методів. Найбільш простим рішенням є зберігання таких даних в окремих (поза базою даних) файлах із заміною “довгих” даних у кортежі на ім’я відповідного файла. У деяких СУБД “довгі” дані зберігалися в окремому наборі сторінок зовнішньої пам'яті, які були поєднані фізичними посиланнями. Однак такі рішення значно обмежують можливість роботи з “довгими” даними, наприклад, виникають труднощі видалення декількох байтів з середини двомегабайтного рядка. У наш час все частіше використовують метод, коли “довгі” дані мають структуру у вигляді B-дерев послідовностей байтів. Як правило, на одній сторінці даних зберігаються кортежі тільки одного відношення. Однак існує можливість зберігання на одній сторінці кортежів декількох відношень. Таке розміщення потребує додаткових витрат в організації службової інформації (при кожному кортежі потрібно зберігати інформацію про відповідне відношення), але це дозволяє скоротити число обмінів з зовнішньою пам'яттю при виконанні з'єднань;

- **зміна схеми відношення, що зберігається, з додаванням нового стовпця не викликає потреби у фізичній реорганізації відношення.** Достатньо лише змінити інформацію в описувачі відношення та розширити кортежі тільки при занесенні інформації у новий стовпець.

Менш поширеним є **зберігання відношення за стовпцями** (другий підхід). При такій організації зберігання даних сумарно у середньому витрачається менше зовнішньої пам'яті, оскільки дублікати значень не зберігаються. При цьому за один обмін з зовнішньою пам'яттю у загальному випадку прочитується більше корисної інформації. Перевагою такого підходу є можливість використання значень стовпця відношення для оптимізації виконання операцій з'єднання. Однак при цьому потрібні істотні додаткові дії для збирання цілого кортежу.

Поширеним способом підвищення ефективності СУБД є кластеризація відношення за значеннями одного або декількох стовпців. Корисною для оптимізації з'єднань є сумісна кластеріизація декількох відношень.

З метою використання можливостей розпаралелювання обмінів з зовнішньою пам'яттю іноді застосовують схему декластеризованого зберігання відношень: кортежі із загальним значенням стовпця декластеризації розміщують на різних дискових пристроях, обміни з якими можна виконувати паралельно.

10.2 Методи організації індексів

Основне призначення індексів полягає у забезпеченні ефективного прямого доступу до кортежу відношення за ключем. Як правило, індекс визначається для одного відношення, і ключем є значення атрибута.

Якщо ключем індексу є можливий ключ відношення, то індекс має бути **унікальним**, тобто не містити дублікатів ключа. Але на практиці, як правило, при визначенні первинного ключа відношення автоматично заводить індекс, а єдиним способом визначення можливого ключа, відмінного від первинного, є явне створення унікального індексу. Це пов'язано з тим, що для перевірки збереження властивості унікальності ключа необхідним є індексне підтримання.

Корисною властивістю індексу, поряд з його унікальністю, є забезпечення **послідовного перегляду кортежів відношення** у діапазоні значень ключа в порядку їх зростання або зменшення. Це пов'язано з тим, що при виконанні багатьох операцій мовного рівня є потрібним сортування відношень відповідно до значень деяких атрибутів.

Крім того, для оптимізації виконання операції еквіз'єднання відношень застосовують так звані **мультиіндекси** для декількох відношень, які мають загальні атрибути. Будь-який з цих атрибутів (або їх набір) може бути ключем мультиіндекса. Значенню ключа відповідає набір кортежів усіх зв'язаних мультиіндексом відношень, значення указаних атрибутів яких збігаються зі значенням ключа.

Необхідно зазначити, що загальною ідеєю будь-якого створення індексів є зберігання впорядкованого списку значень ключа з прив'язкою до кожного значення ключа списку ідентифікаторів кортежів. Методи організації індексів відрізняються один від одного в основному способом пошуку ключа із заданим значенням. Коротко розглянемо суть деяких методів організації індексів.

Метод B-дерева. Як правило, логічно B-дерево являє собою збалансоване сильно гіллясте дерево у зовнішній пам'яті. Логічну структуру B-дерева показано на рисунку 10.2.

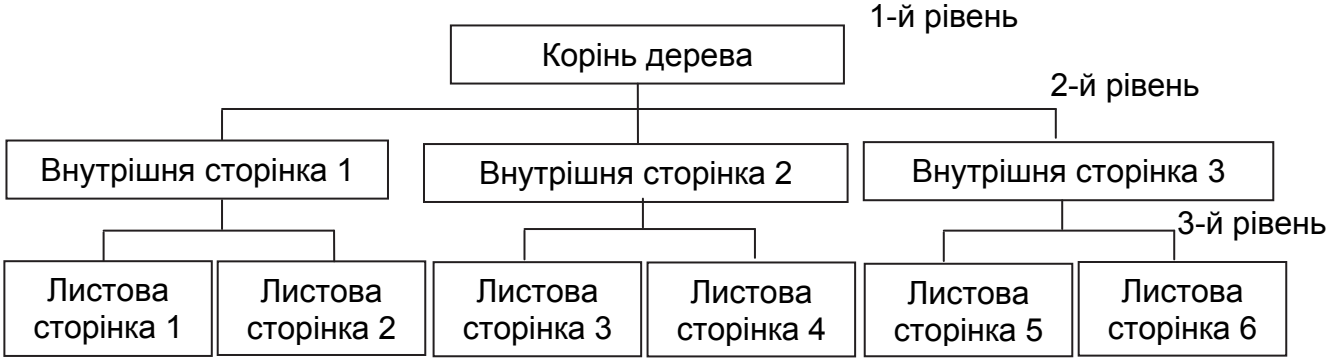


Рисунок 10.2 – Логічна структура B-дерева

Збалансованість означає, що довжина шляху від кореня дерева до будь-якого його листа є однаковою.

Гіллястість дерева – це властивість кожного вузла дерева посилатися на велике число вузлів-нащадків.

Фізично B-дерево створюється як мультисписочна структура сторінок зовнішньої пам'яті, тобто кожному вузлу дерева відповідає блок зовнішньої пам'яті (сторінка). Внутрішні та листові сторінки зазвичай мають різну структуру.

У типовому випадку структуру внутрішньої сторінки B-дерева показано на рисунку 10.3.

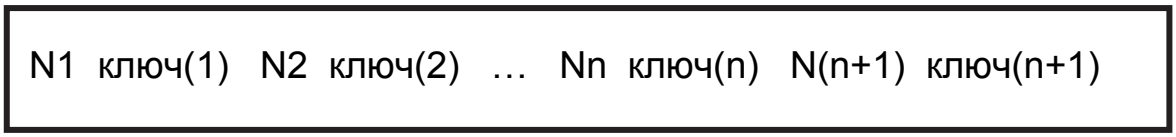


Рисунок 10.3 – Структура внутрішньої сторінки B-дерева

На рисунку 10.3 позначено N1, N2, …, N(n+1) – номера листових сторінок, а ключ(1), ключ(2), ..., ключ(n+1) – значення ключа.

При цьому внутрішня сторінка має такі властивості:

- ключ(1) ≤ ключ(2) ≤ ... ≤ ключ(n);
- у сторінці дерева Nm знаходяться ключі k зі значеннями ключ(*m*) ≤ *k* ≤ ключ(*m*+1).

Листова сторінка B-дерева, як правило, має іншу структуру, яку показано на рисунку 10.4.

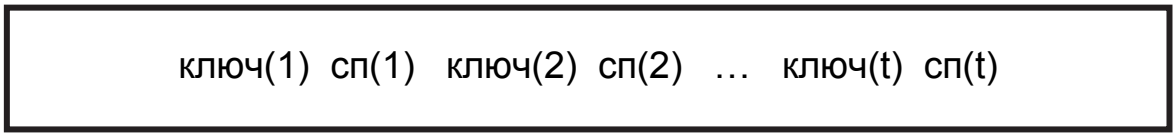


Рисунок 10.4 – Структура листової сторінки B-дерева

На рисунку 10.4 позначено ключ(1), ключ(2), ..., ключ(t) – значення ключа, а сп(1), сп(2), …, сп(t) – список ідентифікаторів кортежів (tid).

При цьому листова сторінка має такі властивості:

- ключ(1) < ключ(2) < ... < ключ(t);
- сп(t) – впорядкований список ідентифікаторів кортежів (tid), що містить значення ключ(t);
- листові сторінки, пов'язані одно- або двонаправленим списком.

Коротко розглянемо, як відбуваються деякі операції у B-дереві.

Пошук у B-дереві – це проходження від кореня дерева до листової сторінки відповідно до заданого значення ключа. Оскільки дерево дуже гіллясте та збалансоване, то для виконання пошуку за будь-яким значенням ключа буде потрібно одне і те ж число обмінів із зовнішньою пам'яттю. При цьому у B-дереві автоматично підтримується властивість збалансованості. Нижче показано, як це здійснюється під час виконання операцій занесення та видалення записів.

Занесення нового запису у B-дерево. Спочатку проводиться **пошук листової сторінки,** тобто пошук сторінки за ключем. Якщо у B-дереві немає ключа із заданим значенням, то буде створено номер

сторінки, на якій його розмістять, а також будуть визначені відповідні координати усередині сторінки.

Далі здійснюється **розміщення запису на місце.** Ця операція проводиться у буферах оперативної пам'яті. Листова сторінка, в яку потрібно занести запис, прочитується у буфер, і у ньому виконується операція вставки. Розмір буфера має перевищувати розмір сторінки зовнішньої пам'яті.

Якщо після виконання вставки нового запису, розмір частини буфера, що був використаний, не перевершує розміру сторінки, то виконання операції занесення нового запису закінчується. Після цього буфер негайно виштовхується у зовнішню пам'ять або тимчасово зберігається в оперативній пам'яті залежно від політики управління буферами.

У разі переповнювання буфера, тобто якщо розмір його частини, що була використана, перевищує розмір сторінки, то сторінка поділяється на дві. Для цього запрошується нова сторінка зовнішньої пам'яті. Потім частина буфера, що використовується, розбивається навпіл так, щоб друга половина також починалася з ключа. Після цього друга половина записується у знов виділену сторінку, а у старій сторінці модифікується значення розміру вільної пам'яті. Наприкінці модифікуються посилання за списком листових сторінок.

Необхідно зазначити, що для забезпечення доступу від кореня дерева до нової заведеної сторінки, необхідно відповідним чином модифікувати внутрішню сторінку, що є предком раніше застосованої листової сторінки. Отже необхідно вставити у нову сторінку відповідне значення ключа та посилання на нову сторінку. При виконанні цієї операції може знову відбутися переповнювання тепер вже внутрішньої сторінки, і її буде поділено на дві. В результаті буде потрібним значення ключа та посилання на нову сторінку вставити у внутрішню сторінку-предок, що є вищою за ієрархією і т.д.

Граничним випадком є переповнювання кореневої сторінки B-дерева. У цьому випадку вона також поділяється на дві, і створюється нова коренева сторінка дерева, тобто його глибина збільшується на одиницю.

Видалення запису. Спочатку здійснюється **пошук запису за ключем.** Якщо запис не знайдено, то видаляти нічого не потрібно і операція закінчується.

У разі **реального видалення запису** ця операція проводиться у буфері, в який поміщено (зчитано) відповідну листову сторінку. Якщо після виконання цієї операції розмір зайнятої у буфері області виявляється таким, що його сума з розміром зайнятої області у листових сторінках, що розміщені зліва або справа від даної сторінки, є більшим, ніж розмір сторінки, операція завершується.

Якщо цей розмір менше, то проводиться злиття з правою або лівою сторінкою, тобто у буфері створюється новий образ сторінки, що містить загальну інформацію з даної сторінки та сторінки, що розташована або

зліва, або справа. Листова сторінка, яка стала непотрібною (вільною від інформації), заноситься у список вільних сторінок. Після цього відповідним чином коректується список листових сторінок.

Необхідно зазначити, що для усунення можливості доступу від кореня до звільненої сторінки, потрібно видалити відповідне значення ключа та посилання на звільнену сторінку з внутрішньої сторінки – її предка. При цьому може виникнути потреба у злитті цієї сторінки зі сторінкою, що розташовується зліва або справа і т.д.

Граничним випадком є повне спустошення кореневої сторінки дерева, яке можливе після злиття останніх двох нащадків кореня. В цьому випадку коренева сторінка звільняється, а глибина дерева зменшується на одиницю.

Перевагою такого методу створення індексів є те, що при виконанні операцій вставки та видалення властивість збалансованості B-дерева зберігається, а зовнішня пам'ять витрачається достатньо економно.

Головною проблемою цього методу є те, що при виконанні операцій модифікації дуже часто можуть виникати розбивання та злиття сторінок.

Для того, щоб мінімізувати їх число застосовуються:

- попереджуючі розбивання сторінок, тобто розбивання сторінки не у разі її переповнювання, а декілька раніше, коли заповненість сторінки досягає деякого рівня;
- переливання, тобто підтримування рівноважного заповнення сусідніх сторінок;
- злиття трьох сторінок у дві, тобто породження двох листових сторінок на основі вмісту трьох сусідніх.

Метод хешірування. Цей метод є альтернативним методу B-дерева та стає все більш популярним. Загальною ідеєю методу хешірування є застосування до значення ключа деякої функції згортки (хеш-функції), що створює значення меншого розміру. Згортка значення ключа потім використовується для доступу до запису.

У найпростішому, класичному випадку, згортка ключа використовується як адреса у таблиці, що містить ключі та записи. Основною вимогою до хеш-функції є рівномірний розподіл значень згортки. При виникненні колізій, тобто коли виникає одна і та ж згортка для декількох значень ключа, утворюються ланцюжки переповнювання.

Головним обмеженням цього методу є фіксований розмір таблиці. Якщо таблиця заповнена дуже сильно або переповнена, то виникне дуже багато ланцюжків переповнювання, і головна перевага хешірування – доступ до запису за одне звернення до таблиці втрачається. Розширення таблиці потребує її повного перероблення на основі нової хеш-функції зі значенням згортки більшого розміру. Однак це може потребувати багато часу і для баз даних є абсолютно неприйнятним.

Для вирішення цієї проблеми, як правило, вводять проміжні таблиці-довідники, що містять значення ключів та адреси записів, а самі записи

зберігаються окремо. Тоді у разі переповнювання довідника є потрібним тільки його перероблення, що приводить до менших накладних витрат. Щоб уникнути необхідності повного перероблення довідників, при їх створенні часто використовують техніку B-дерев з розбиванням та злиттям. При цьому хеш-функція змінюється динамічно залежно від глибини B-дерева. В цілому методи B-дерев та хешірування все більш зближуються.

10.3 Зберігання журнальної та службової інформації

Журнальна інформація. Порядок збереження журнальної інформації та структура журналу залежать від конкретної реалізації СУБД. Журнал, як правило, являє собою послідовний файл із записами змінного розміру, які можна переглядати у прямому або зворотному порядку. Обміни проводяться стандартними порціями (сторінками) з використанням буфера оперативної пам'яті. У правильно організованих системах структура журнальних записів є відомою тільки компонентам СУБД, які відповідають за журналізацію та відновлення.

До ведення файла-журналу ставлять особливі вимоги, що стосуються надійності зберігання БД. Це пов’язано з тим, що наповнення та коректність даних у журналі є важливими під час відновлення бази даних після збоїв. Як правило, у надійних БД підтримуються дві ідентичні копії журналу на різних пристроях зовнішньої пам'яті.

Службова інформація. Службова інформація призначена для коректної роботи підсистеми управління даними у зовнішній пам'яті. Для забезпечення роботи цієї підсистеми необхідно підтримувати інформацію, яка використовується тільки цією підсистемою і яку не видно за допомогою підсистеми мовного рівня. Набір структур службової інформації залежить від загальної організації системи. Найбільш часто у розвинених СУБД підтримуються такі службові дані: внутрішні каталоги; описувачі вільної та зайнятої пам'яті у сторінках відношення; скріплення сторінок одного відношення.

За допомогою **внутрішніх каталогів** наводяться фізичні властивості об'єктів бази даних, наприклад, кількість атрибутів відношення, їх розміри, а також типи даних та опис індексів, визначених для даного відношення і т.д.

Описувачі вільної та зайнятої пам'яті у сторінках відношення призначені для знаходження вільного місця під час занесення кортежу до відношення. Виникають окремі проблеми, які потрібно вирішувати. Ці проблеми є пов’язаними із задаванням пошуку вільного місця у випадках некластеризованих і кластеризованих відношень (додатково використовують кластеризований індекс), а також із звільненням сторінки в умовах мультидоступу.

Проблема **скріплення сторінок одного відношення** виникає, якщо в одному файлі зовнішньої пам'яті розташовуються сторінки декількох відношень. У такій ситуації необхідно з'єднувати сторінки одного відношення.

Як правило, у цьому випадку використовують прямі посилання між сторінками, але це часто призводить до ускладнень, які виникають під час синхронізації транзакцій. Наприклад, при застосуванні такого способу з’єднання сторінок відношень стає важко звільняти та заводити нові сторінки відношення. Для усунення цього недоліку часто використовують непряме скріплення сторінок із застосуванням службових індексів. Наприклад, використовують механізм для опису вільної пам'яті й скріплення сторінок на основі B-дерев.

Запитання для самоперевірки

1. Які особливості реляційних СУБД впливають на побудову зовнішньої пам'яті ?
2. Якими позитивними та негативними якостями відрізняються зберігання відношень покортежно та за стовпцями ?
3. Які властивості має B-дерево у зовнішній пам'яті та що вони характеризують ?
4. У чому полягає різниця між внутрішніми та листовими сторінками ?
5. Як підтримується збалансованість B-дерева ?
6. Як здійснюється збереження журнальної інформації у реляційних СУБД ?
7. З якою метою в реляційних СУБД підтримується службова інформація ?

Лекція № 11

УПРАВЛІННЯ ТРАНЗАКЦІЯМИ

Навчальні цілі:
- розглянути механізм управління транзакціями;
- вивчити методи серіалізації транзакцій.

Навчальні питання:
1. Механізм управління транзакціями.
2. Методи серіалізації транзакцій.

11.1 Механізм управління транзакціями

Підтримання механізму транзакцій є основним показником рівня вдосконаленості СУБД і основою забезпечення цілісності баз даних. На базі цього механізму створюється основа для забезпечення ізольованості користувачів у багатокористувальних системах. Ці два аспекти механізму транзакцій є взаємозв'язаними.

Розглянемо поняття транзакції і застосування механізму транзакції під час забезпечення цілісності баз даних та ізольованості користувачів у багатокористувальних системах.

Транзакція являє собою неподільну щодо впливу на базу даних послідовність операторів маніпулювання даними, при виконанні якої результати всіх операторів, що входять у транзакцію, або відображаються (фіксуються), або повністю є відсутніми у базі даних. Нагадаємо, що до операцій маніпулювання даними належать: зчитування, видалення, вставлення та модифікація даних.

Якщо транзакція завершується оператором **COMMIT** (зафіксувати), то результати дії всіх операторів, що входять у транзакцію, гарантовано фіксуються у зовнішній пам'яті. Якщо транзакція завершується оператором **ROLLBACK** (ліквідувати), то результати дії всіх операторів, що входять у транзакцію, гарантовано є відсутніми у зовнішній пам'яті.

Поняття транзакції безпосередньо пов'язано з поняттям **цілісності баз даних.** Під час маніпулювання даними в реляційних БД можуть існувати такі обмеження цілісності, які просто неможливо не порушити при дії тільки одного оператора зміни БД. Тому для підтримання таких обмежень цілісності допускається їх порушення всередині транзакції з тією умовою, що до моменту завершення транзакції умови цілісності будуть дотримані. Отже, у системах з розвиненими засобами обмеження та контролю цілісності **кожна транзакція починається при цілісному стані БД і має залишити цей стан цілісним після свого завершення**. Недотримання цієї умови призводить до того, що замість фіксації результатів транзакції відбувається її відкат, тобто замість оператора

COMMIT виконується оператор **ROLLBACK**, і БД залишається в такому стані, в якому знаходилася до початку транзакції, тобто у цілісному стані.

Розрізняють два види обмежень цілісності: що негайно перевіряються і що відкладаються.

До обмежень цілісності, що **негайно перевіряються,** належать такі обмеження, перевірку яких безглуздо або навіть неможливо відкладати. Наприклад, обмеженням, перевірку якого відкладати безглуздо, є обмеження домену – вік співробітника не може перевищувати 150 років. Обмеження цілісності, що негайно перевіряються, відповідають рівню окремих операторів мовного рівня СУБД. При їх порушеннях не проводиться відкат транзакції, а лише відкидається відповідний оператор.

Обмеження цілісності, що **відкладаються**, – це обмеження бази даних, а не окремих операцій. Такі обмеження перевіряються наприкінці виконання транзакції, та їх порушення потребує автоматичної заміни оператора **COMMIT** на оператор **ROLLBACK**. Отже, у момент завершення транзакції перевіряються всі обмеження цілісності, що відкладаються, які визначені у БД. Під час реалізації конкретної БД намагаються динамічно визначити ті обмеження цілісності, які можуть бути порушені при виконанні тієї чи іншої транзакції.

У багатокористувальних системах з однією базою даних одночасно можуть працювати декілька користувачів або прикладних програм. Граничним завданням такої системи є забезпечення **ізольованості користувачів**, тобто створення достовірної та надійної ілюзії того, що кожен з користувачів працює з базою даних один.

Для забезпечення ізольованості користувачів, також як і для збереження цілісності БД, використовується механізм транзакції. Дійсно, якщо кожному сеансу роботи з базою даних поставити у відповідність одну транзакцію, то кожен користувач буде починати роботу з узгодженим станом бази даних та закінчувати роботу з БД у тому ж стані, тобто у такому стані, в якому база даних могла б знаходитися, навіть якщо б користувач працював з нею один.

При дотриманні обов'язкової вимоги підтримання цілісності бази даних розглядають три основних рівня ізольованості транзакцій.

Перший рівень – відсутність втрачених змін.

Один із сценаріїв сумісного виконання двох транзакцій (**Тр.1** і **Тр.2**) може бути таким:

- **Тр.1** змінює об'єкт бази даних ***A***;
- до завершення **Тр.1** транзакція **Тр.2** також змінює об'єкт ***A***;
- **Тр.2** завершується оператором **ROLLBACK** (наприклад, внаслідок порушення обмежень цілісності).

У результаті виконання цього сценарію під час повторного читання об'єкта ***A*** користувач не бачить змін цього об'єкта, які були проведені **Тр.1** раніше. Така ситуація називається **ситуацією втрачених змін**, яка суперечить вимозі ізольованості користувачів. Щоб її уникнути необхідно,

щоб до завершення **Тр.1** ніяка інша транзакція не могла змінювати об'єкт ***A***. Цей рівень є мінімальною вимогою до СУБД щодо синхронізації транзакцій, які виконуються паралельно.

Другий рівень – відсутність читання «брудних даних».

Другий сценарій сумісного виконання двох транзакцій (**Тр.1** і **Тр.2**) може бути таким:

- **Тр.1** змінює об'єкт бази даних ***A***;
- у той же час **Тр.2** зчитує об'єкт ***A***.

Під час виконання цього сценарію **Тр.1**, що змінює об'єкт, ще не завершена, в результаті чого **Тр.2** «бачить» неузгоджені «брудні» дані (зокрема, операція **Тр.1** може бути відкинутою). Така ситуація також не відповідає вимозі ізольованості користувачів, тобто кожен користувач починає свою транзакцію при узгодженому стані бази даних та бажає бачити узгоджені дані. Щоб уникнути ситуації читання «брудних» даних, необхідно, щоб до завершення **Тр.1**, що змінює об'єкт ***A***, ніяка інша транзакція не зчитувала об'єкт ***A***. Мінімальною вимогою є блокування зчитування об'єкта ***A*** до завершення операції його зміни у **Тр.1**.

Третій рівень – відсутність читань, що не повторюються.

Ще одним сценарієм сумісного виконання двох транзакцій (**Тр.1** і **Тр.2**) може бути такий:

- **Тр.1** зчитує об'єкт бази даних ***A***;
- до завершення **Тр.1** транзакція **Тр.2** змінює об'єкт ***A*** і успішно завершується оператором **COMMIT**;
- **Тр.1** повторно зчитує об'єкт ***A*** і бачить його змінений стан.

Щоб уникнути читань, що не повторюються, необхідно, щоб до завершення **Тр.1** ніяка інша транзакція не змінювала об'єкт ***A***. Як правило, цей рівень є максимальним щодо синхронізації транзакцій, які виконуються паралельно.

Необхідно зазначити, що існує можливість забезпечення різних рівнів ізольованості для різних транзакцій, що виконуються в одній системі баз даних. Існують також додатки, для яких є достатнім перший рівень підтримання цілісності, наприклад, прикладні або системні статистичні утиліти, для яких некоректність індивідуальних даних є неважливою. При цьому вдається істотно скоротити накладні витрати СУБД та підвищити загальну ефективність.

У деяких випадках також може виникати проблема ізольованості транзакцій, яка пов’язана з виникненням **кортежів-фантомів**.

Розглянемо сценарій сумісного виконання двох транзакцій (**Тр.1** і **Тр.2**), під час якого може виникнути така ситуація:

- **Тр.1** виконує оператор ***A***, який здійснює вибір кортежів відношення ***R*** за умовою вибору ***S***, тобто вибирається частина кортежів відношення **R**, які задовольняють умові ***S***;
- до завершення **Тр.1** транзакція **Тр.2** вводить у відношення ***R*** новий кортеж ***r***, що задовольняє умові ***S***, та успішно завершується;

- **Тр.1** повторно виконує оператор ***A***, і в результаті з'являється кортеж, який був відсутнім при першому виконанні оператора ***A***.

Така ситуація не відповідає вимозі ізольованості користувачів і може виникнути навіть на третьому рівні ізольованості транзакцій. Щоб уникнути появи кортежів-фантомів, є потрібним вищий «логічний» рівень синхронізації транзакцій, наприклад, предикативні синхронізаційні захоплення.

11.2 Методи серіалізації транзакцій

Серіалізація транзакцій – це механізм їх виконання за деяким серіальним планом. Забезпечення такого механізму є основною функцією компонента СУБД, який відповідає за управління транзакціями.

План (спосіб) виконання набору транзакцій називається **серіальним**, якщо результат сумісного виконання транзакцій є еквівалентним результату деякого послідовного виконання цих же транзакцій.

У наш час є два основних підходи до серіалізації транзакцій. Перший з них основано на синхронізаційних захопленнях об'єктів бази даних, а другий – використанні часових міток. Обидва ці підходи призначені для виявлення конфліктів транзакцій та їх усунення.

Нагадаємо, що під час роботи з БД можуть виникати такі конфлікти транзакцій:

- **W-W – Тр.2** намагається змінювати об'єкт, який був змінено **Тр.1**;
- **R-W – Тр.2**, яка ще не закінчилася, намагається змінювати об'єкт, який було зчитано **Тр.1**;
- **W-R – Тр.2**, яка ще не закінчилася, намагається зчитувати об'єкт, який був змінено **Тр.1**, що не закінчилася.

Найбільш часто у реляційних СУБД використовують перший підхід, що основується на дотриманні двофазного протоколу синхронізаційних захоплень об'єктів БД.

Сформулюємо **двофазний протокол (2PL) синхронізаційних захоплень об'єктів БД**. Перед виконанням будь-якої операції у транзакції з об'єктом ***r*** бази даних від імені цієї транзакції запрошується синхронізаційне захоплення об'єкта ***r*** у відповідному режимі (режим залежить від виду операції).

Основними режимами синхронізаційних захоплень є:

- **сумісний режим** – **S** (Shared), який означає роздільне захоплення об'єкта, таке захоплення є необхідним для виконання операції зчитування об'єкта;
- **монопольний режим** – **X** (eXclusive), який означає монопольне захоплення об'єкта, таке захоплення є необхідним для виконання операцій занесення, видалення та модифікації.

Слід зазначити, що захоплення об'єктів декількома транзакціями під час зчитування даних є сумісними, тобто декільком транзакціям дозволяється зчитувати один і той же об'єкт.

У той же час захоплення об'єкта однією транзакцією для зчитування не є сумісним із захопленням іншою транзакцією того ж об'єкта для внесення (запису) даних. Крім того, захоплення одного об'єкта різними транзакціями для внесення (запису) даних є несумісними.

Правила сумісності захоплень одного об'єкта різними транзакціями можна подати у вигляді таблиці 11.1.

Таблиця 11.1 – Правила сумісності захоплень одного об'єкта різними транзакціями

Режими захоплень	**X**	**S**
–	так	так
X	ні	ні
S	ні	так

У першому стовпці таблиці наведені можливі режими, в яких може заходитися об'єкт до початку синхронізаційних захоплень. При цьому тире відповідає стану об'єкта, для якого раніше не було встановлено ніяке захоплення. У другому та третьому стовпцях таблиці зазначені можливості сумісних захоплень у різних режимах. Слово «так» означає, що режими є сумісними, а слово «ні» – несумісними. Необхідно зазначити, що слово «ні» у таблиці відповідає можливим випадкам конфліктів транзакцій щодо доступу до об'єктів бази даних (**W-W**, **R-W**, **W-R**), які були розглянуті вище. Із таблиці 11.1 видно, що сумісність **S** захоплень відповідає тому, що конфлікту **R-R** не існує.

Транзакція, що виконала запит синхронізаційного захоплення об'єкта БД, який раніше був вже захоплений іншою транзакцією у несумісному режимі, блокується доти, доки захоплення з цього об'єкта не буде знято.

Таким чином, для забезпечення серіалізації транзакцій на третьому рівні ізольованості (відсутність читань, що не повторюються) синхронізаційні захоплення об'єктів, які були проведені за ініціативою транзакції, можна знімати тільки при її завершенні. При цьому відповідно до двофазного протоколу синхронізаційних захоплень 2PL виконання транзакції розбивається на дві фази:

- перша фаза транзакції – накопичення захоплень;
- друга фаза (фіксація або відкат) – звільнення захоплень.

Під час здійснення синхронізаційних захоплень виникає основна проблема – що необхідно вважати об'єктом захоплення.

Як об'єкти захоплення у реляційних базах даних можуть бути:

- **файл** – фізичний об'єкт відносно бази даних, тобто область зберігання декількох відношень, а також, можливо, індексів;

- **відношення** – логічний об'єкт, який відповідає безлічі кортежів даного відношення;

- **сторінка даних** – фізичний об'єкт, що зберігає кортежі одного або декількох відношень, індексну або службову інформацію;

- **кортеж** – елементарний логічний об'єкт бази даних.

Необхідно зазначити, що під час виконання будь-яких операцій з об'єктами бази даних, операція з кортежем (нижнім рівнем об’єкта БД) фактично є операцією зі сторінкою, в якій цей кортеж зберігається (наступним рівнем об’єкта БД), а також з відповідним відношенням (перед верхнім рівнем об’єкта БД) і з файлом, що містить відношення (верхнім рівнем об’єкта БД). Тому перед проведенням синхронізаційних захоплень необхідно визначитися (вибрати) рівень об'єкта захоплення.

Виходячи з цього можна зазначити, що чим вище знаходиться об'єкт синхронізаційного захоплення (неважливо, якої природи цей об'єкт – логічний або фізичний), тим менше синхронізаційних захоплень буде підтримуватися у системі. Це, у свою чергу, приведе до того, що накладні витрати будуть меншими. Крім того, якщо як об'єкт синхронізаційного захоплення вибрати файл або відношення, то буде вирішена навіть проблема кортежів-фантомів.

У той же час, якщо використовувати верхні рівні для синхронізаційних захоплень об'єктів, то зростає ймовірність конфліктів транзакцій. Це, у свою чергу, приведе до того, що буде зменшена допустима ступінь їх паралельного виконання. Отже, якщо користувач збільшує рівень об'єкта синхронізаційного захоплення, то він навмисно робить ситуацію «грубою», що призводить до конфліктів у тих ситуаціях, коли насправді конфліктів немає.

У більшості сучасних СУБД використовуються синхронізаційні захоплення кортежів. Однак при цьому може виникнути ситуація, коли потрібно виконати операцію знищення відношення, а об’єктом синхронізаційного захоплення є кортеж. Перед виконанням такої операції необхідно буде провести захоплення всіх існуючих кортежів відношення, а це буде збільшувати накладні витрати. Крім того, це не запобігає можливості паралельної вставки в іншій транзакції нового кортежу у відношення, що знищується.

Для усунення таких недоліків було розроблено апарат **гранульованих синхронізаційних захоплень**.

Під час застосування такого підходу синхронізаційні захоплення можуть запрошуватися до об'єктів різного рівня: до файлів, відношень та кортежів. Необхідний рівень об'єкта визначається тим, яка операція виконується. Наприклад, для виконання операції знищення відношення об'єктом синхронізаційного захоплення має бути все відношення, а для виконання операції видалення кортежу – тільки цей кортеж. Об'єкт будь-якого рівня може бути захопленим у сумісному **S** або монопольному **X** режимах. Для цього застосовується спеціальний протокол гранульованих

захоплень, а також нові типи захоплень. Протокол гранульованих захоплень можна сформулювати так: перед захопленням об'єкта у режимі **S** або **X** відповідний об'єкт більш верхнього рівня має бути захопленим у режимі **IS**, **IX** або **SIX**:

Режим IS (Intented for Shared lock) відносно деякого складного об'єкта (**CO**) означає намір захопити деякий об'єкт (**O**), що входить до складного об’єкта (**CO**), у сумісному режимі. Наприклад, під час наміру зчитувати кортежі з відношення **R** це відношення має бути захопленим у режимі **IS**, а до цього, у свою чергу, у такому ж режимі має бути захопленим файл.

Режим IX (Intented for eXclusive lock) відносно деякого складного об'єкта (**CO**) означає намір захопити деякий об'єкт (**O**), що входить до складного об'єкта (**CO**), у монопольному режимі. Наприклад, під час наміру видалити кортежі з відношення **R** це відношення має бути захопленим у режимі **IX**, а до цього, у свою чергу, у такому ж режимі має бути захопленим файл.

Режим SIX (Shared, Intented for eXclusive lock) відносно деякого складного об'єкта (**CO**) означає сумісне захоплення всього цього об'єкта (**CO**) з наміром згодом захоплювати які-небудь об'єкти (**O**), що входять до нього (**CO**), у монопольному режимі. Наприклад, якщо виконується довга операція переглядання відношення з можливістю видалення деяких кортежів, що переглядаються, то більш економно захопити це відношення у режимі **SIX**, а до цього, у свою чергу, захопити файл у режимі **IS**.

Повні правила сумісності захоплень одного об'єкта різними транзакціями наведено у таблиці 11.2. Позначення у стовпцях таблиці 11.2 є такими ж, як і у таблиці 11.1.

Таблиця 11.2 – Повні правила сумісності захоплень

Режими захоплень	**X**	**S**	**IX**	**IS**	**SIX**
–	так	так	так	так	так
X	ні	ні	ні	ні	ні
S	ні	так	ні	так	ні
IX	ні	ні	так	так	ні
IS	ні	так	так	так	так
SIX	ні	ні	ні	так	ні

Необхідно зазначити, що за допомогою методу гранульованих синхронізаційних захоплень не можна вирішити проблему кортежів-фантомів. Для вирішення цієї проблеми необхідно перейти від захоплень індивідуальних об'єктів бази даних до захоплення умов (предикатів), яким задовольняють ці об'єкти. Такий метод називають методом **предикативних синхронізаційних захоплень.**

Проблема кортежів-фантомів не виникає при використанні для синхронізації рівня відношень саме тому, що відношення як логічний об'єкт є неявною умовою для кортежів, що входять до нього. Захоплення відношення – це простий окремий випадок предикативного захоплення.

У зв’язку з тим, що під час здійснення будь-якої операції з реляційною БД у цій операції указується не конкретний набір об'єктів БД, з якими потрібно виконати операцію, а умова, якій мають задовольняти об'єкти цього набору, то доцільно було б здійснювати синхронізаційні захоплення у режимі **S** або **X** саме цієї умови. Однак умови, що задаються у мовах БД, не дозволяють визначити сумісність двох предикативних захоплень, а без цього використовувати предикативні захоплення для синхронізації транзакцій неможливо.

Сумісність двох предикативних захоплень порівняно легко перевіряється у разі застосування **простих умов**, тобто умов, які являють собою кон'юнкцію простих предикатів, що мають вигляд: ім’я атрибуту { = > < } значення. Отже, у разі типової організації реляційної СУБД прості умови можна використовувати як основу предикативних захоплень.

Розглянемо сумісність двох предикативних захоплень для простих умов на прикладі геометричної інтерпретації, яку показано на рисунку 11.1.

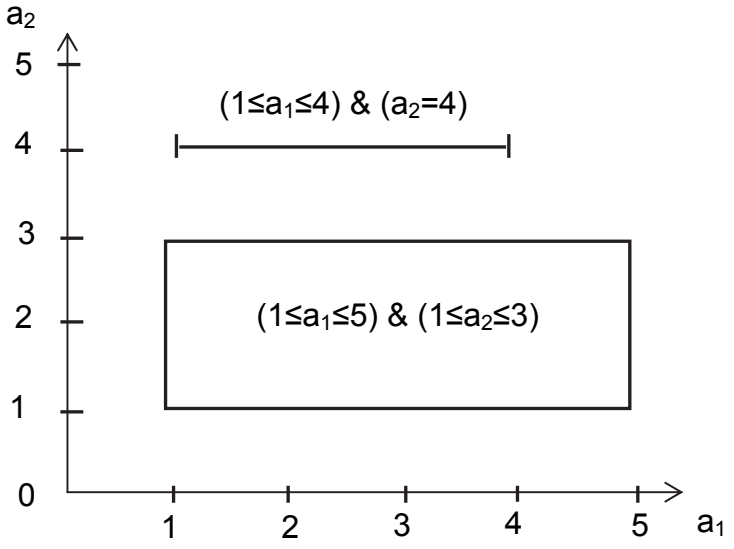


Рисунок 11.1 – Геометрична інтерпретація сумісності двох предикативних захоплень для простих умов

Розглянемо відношення ***R*** з атрибутами $a_1, a_2, ..., a_n$. Нехай $m_1, m_2, ..., m_n$ – безліч допустимих значень атрибутів $a_1, a_2, ..., a_n$. Тоді можна скласти кінцевий n-мірний простір можливих значень кортежів відношення ***R.*** Будь-яка проста умова “вирізає” m-мірний прямокутник у цьому просторі ($m \le n$). Тоді предикативні захоплення **S-X**, **X-S**, **X-X** від різних транзакцій є сумісними, якщо відповідні прямокутники не перетинаються.

Із рисунка 11.1 видно, що в яких би режимах **Тр.1** не ставила вимогу захоплення умови $(1 \le a_1 \le 4)$ & $(a_2 = 5)$, а **Тр.2** – умови $(1 \le a_1 \le 5)$ & $(1 \le a_2 \le 3)$, ці захоплення завжди є сумісними.

Необхідно зазначити, що предикативні захоплення простих умов описуються таблицями, які практично не відрізняються від таблиць традиційних синхронізаторів.

Одним з основних недоліків методу серіалізації транзакцій на основі синхронізаційних захоплень є можливість утворення **тупиків** між транзакціями. Тупик може виникнути при застосуванні будь-якого з розглянутих вище методів синхронізаційних захоплень.

Розглянемо ситуацію, при якій може виникнути тупик між двома транзакціями (**Тр.1** і **Тр.2**):

- **Тр.1** і **Тр.2** встановили монопольні захоплення (**X**) об'єктів **r1** і **r2** відповідно;
- після цього **Тр.1** потребує сумісного захоплення (**S**) **r2**, а **Тр.2** – сумісного захоплення (**S**) **r1**;
- жодна з транзакцій не може продовжуватися, отже, монопольні захоплення **X** не будуть знятими, а сумісні **S** – не будуть дозволеними.

У зв’язку з тим, що під час синхронізаційних захоплень тупики є можливими, і тупикові ситуації самі по собі не вирішуються, то такі тупикові ситуації необхідно виявляти і штучно усувати.

Основою виявлення тупикових ситуацій є побудова графа очікування транзакцій.

Граф очікування транзакцій – це орієнтований дводольний граф, в якому існує два типи вершин – вершини, які відповідають транзакціям, та вершини, які відповідають об'єктам захоплення. У такому графі існує дуга, яка проводиться з вершини-транзакції до вершини-об'єкта, якщо для цієї транзакції існує задоволене захоплення об'єкта. У графі існує дуга з вершини-об'єкта до вершини-транзакції, якщо транзакція чекає задоволення захоплення об'єкта.

У системі буде виникати тупикова ситуація, якщо у графі очікування транзакцій є хоч би один цикл.

Для розпізнавання тупиків періодично будується граф очікування транзакцій, і у цьому графі шукаються цикли. Як правило, для знаходження циклів в орієнтованому графі застосовують техніку його редукції.

Редукція графа полягає в тому, що спочатку з графа очікування видаляються всі дуги, що виходять з вершин-транзакцій, але в які не входять дуги з вершин-об'єктів. Отже, враховуються ситуації, коли транзакції, не чекають задоволення захоплень, а успішно завершуються та звільняють захоплення.

У подальшому для тих вершин-об'єктів, для яких не залишилося вхідних дуг, але існують ті, що виходять, орієнтація дуг, що виходять, змінюється на протилежну. Так моделюється ситуація задоволення захоплень. Після цього знову перевіряється перший крок редукції графа.

Цей процес продовжується доти, доки на першому кроці не припиниться видалення дуг. Якщо після виконання цієї процедури у графі залишилися дуги, то вони обов'язково будуть утворювати цикл.

У разі, якщо було знайдено цикл у графі очікування транзакцій, то необхідно якось забезпечити можливість продовження роботи хоч би для частини транзакцій, що потрапили у тупикову ситуацію. Вихід з цієї ситуації починається з вибору у циклі транзакцій так званої транзакції-жертви, тобто транзакції, якою вирішено пожертвувати, щоб забезпечити можливість продовження роботи інших транзакцій. При цьому критерієм вибору є важливість транзакції, тобто жертвою вибирається найменш важлива транзакція. У свою чергу важливість транзакції визначається на основі багатофакторного оцінювання. До основних факторів, що мають різну вагу, належать час виконання, кількість накопичених захоплень, пріоритет.

Після вибору транзакції-жертви виконується повний або частковий відкат цієї транзакції. У результаті виконання цієї процедури звільняються захоплення і продовжується виконання інших транзакцій.

Платою за таке насильницьке усунення тупикових ситуацій є порушення принципу ізольованості користувачів, якого не можна уникнути. Ще одним недоліком такого підходу є те, що у по-справжньому розподілених СУБД вартість побудови графа очікування стає дуже великою. Це пов’язано з тим, що у таких СУБД транзакції можуть виконуватися у різних вузлах мережі. У таких системах, як правило, застосовуються інші методи серіалізації транзакцій.

Необхідно зазначити, що в деяких СУБД (наприклад, в Oracle) для мінімізації кількості конфліктів між транзакціями монопольне захоплення об'єкта блокує тільки транзакції, які змінюються. Після виконання операції модифікації попередня версія об'єкта залишається доступною для зчитування в інших транзакціях. Короткочасне блокування зчитування є потрібним тільки на період фіксації транзакції, яка змінюється, коли оновлені об'єкти стають поточними.

Ще одним методом серіалізації транзакцій, який ефективно працює в умовах рідкісних конфліктів транзакцій і не потребує побудови графа очікування транзакцій, є метод, оснований на використанні часових міток.

Основною ідеєю **методу часових міток** є таке: якщо **Тр.1** почалася раніше **Тр.2**, то система забезпечує такий режим виконання, начебто **Тр.1** була цілком виконана до початку **Тр.2**.

Для цього кожній транзакції надається часова мітка **t**, яка відповідає початку цієї транзакції. Під час виконання операції з об'єктом транзакція позначає його своєю часовою міткою і типом операції (зчитування або зміна).

Перед початком виконання операції **r** з об'єктом транзакція **Тр.2** виконує такі дії:

- перевіряє, чи не закінчилася **Тр.1**, яка позначила цей об'єкт;

- якщо **Тр.1** закінчилася, то **Тр.2** позначає цей об'єкт і виконує свою операцію;

- якщо **Тр.1** не закінчилася, то **Тр.2** перевіряє конфліктність операцій;

- якщо операції є неконфліктними (зчитування), при об'єкті залишається або проставляється часова мітка з меншим значенням, і **Тр.2** виконує свою операцію;

- якщо операції **Тр.2** і **Тр.1** конфліктують, то якщо **t(Тр.2) > t(Тр.1)**, тобто транзакція **Тр.1** є «молодшою», ніж **Тр.2**, проводиться відкат **Тр.1**, а **Тр.2** продовжує роботу;

- якщо **t(Тр.2) < t(Тр.1)**, тобто **Тр.1** «старіша», ніж **Тр.2**, то **Тр.1** отримує нову часову мітку і починається процес наново.

До недоліків методу належать потенційно частіші відкати транзакцій, чим у разі використання синхронізаційних захоплень. Це пов'язано з тим, що конфліктність транзакцій визначається більш грубим способом. Крім того, у розподілених системах не дуже просто створювати глобальні часові мітки з відношенням повного порядку. Однак ці недоліки компенсуються тим, що не потрібно розпізнавати тупикові ситуації, а як зазначалося вище, побудова графа очікування транзакції у розподілених системах має велику вартість.

Запитання для самоперевірки

1. У чому полягає механізм управління транзакціями ?
2. Як застосовується механізм управління транзакціями під час забезпечення цілісності баз даних ?
3. У чому полягає різниця між основними рівнями ізольованості транзакцій ?
4. Які об'єкти захоплення можуть бути у реляційних базах даних ?
5. З якою метою застосовують апарат гранульованих синхронізаційних захоплень ?
6. Як виявляються і усуваються тупикові ситуації під час синхронізаційних захоплень транзакцій ?
7. У чому полягають переваги та недоліки методу часових міток ?

Лекція № 12

ЖУРНАЛІЗАЦІЯ ЗМІН БАЗИ ДАНИХ

Навчальні цілі:
- розглянути основні поняття журналізації змін БД;
- вивчити дії СУБД під час відновлення бази даних після збоїв.

Навчальні питання:
1. Поняття журналізації змін БД.
2. Відновлення бази даних після збоїв.

12.1 Поняття журналізації змін БД

Журналізація спрямована на забезпечення надійності зберігання даних у зовнішній пам'яті, а журналом є особлива частина БД, недоступна користувачам СУБД, в яку надходять записи про всі зміни основної частини БД. У всіх видах журналів застосовується стратегія "попереджувального" запису, тобто реалізується протокол WAL, коли запис про зміну будь-якого об'єкта БД потрапляє у зовнішню пам'ять журналу раніше, ніж змінений об'єкт потрапляє у зовнішню пам'ять основної частини БД.

Таким чином, загальною метою журналізації змін баз даних є забезпечення можливості відновлення узгодженого стану бази даних після будь-якого збою.

До загальних принципів відновлення БД належать:

- результати зафіксованих транзакцій мають бути збереженими у відновленому стані бази даних;
- результати незафіксованих транзакцій мають бути відсутніми у відновленому стані бази даних.

Реалізація цих принципів забезпечує відновлення останнього за часом узгодженого стану бази даних.

Можливими є такі ситуації, при яких потрібно проводити відновлення стану бази даних:

- індивідуальний відкат транзакції;
- відновлення після раптової втрати вмісту оперативної пам'яті (м'який збій);
- відновлення після поломки основного зовнішнього носія бази даних (жорсткий збій).

Найчастіше індивідуальний відкат транзакції здійснюється, коли вона явно завершується оператором **ROLLBACK**. Можливими також є ситуації, коли відкат транзакції ініціюється системою. Наприклад, можуть виникати виняткові ситуації у прикладній програмі (ділення на нуль) або транзакція вибирається як жертва під час усунення тупикової ситуації при

синхронізаційному захопленні. Для відновлення узгодженого стану бази даних при індивідуальному відкаті транзакції необхідно усунути наслідки дії операторів, що модифікували базу даних у цій транзакції.

Ситуація відновлення після раптової втрати вмісту оперативної пам'яті може виникнути при аварійному вимкненні електричного живлення, при виникненні неусувного збою процесора, наприклад, спрацьовуванні контролю оперативної пам'яті і т.д. Під час такого збою втрачається частина бази даних, яка до моменту збою містилася у буферах оперативної пам'яті.

Ситуація відновлення після поломки основного зовнішнього носія бази даних трапляється нечасто, але розвинена СУБД має бути у змозі відновити базу даних у цьому випадку за допомогою архівної копії та журналу змін бази даних. Основою відновлення БД є надмірне зберігання даних у журналі, який містить послідовні записи про зміну бази даних.

У наш час найчастіше використовують два основних варіанти ведення журнальної інформації. У першому варіанті для кожної транзакції створюється окремий локальний журнал змін бази даних, які проводить ця транзакція. Такі локальні журнали використовуються для проведення індивідуальних відкатів транзакцій і можуть зберігатися в оперативній пам'яті. Крім того, створюється загальний журнал змін бази даних, який використовується для відновлення стану БД після м'яких і жорстких збоїв. Основним недоліком цього варіанту є дублювання інформації у локальних і загальному журналах, але він дозволяє швидко проводити індивідуальні відкати транзакцій.

У другому варіанті створюється тільки загальний журнал змін бази даних, який використовується і для проведення індивідуальних відкатів. Такий варіант застосовується найчастіше.

Журналізація змін БД також пов'язана з буферизацією сторінок БД в оперативній пам'яті. Якби запис про зміну бази даних, який має надійти у журнал при виконанні будь-якої операції модифікації бази даних, негайно записувався б і у зовнішню пам'ять, то це б привело до істотного уповільнення роботи системи. Тому записи у журналі також буферизуються: при нормальній роботі чергова сторінка виштовхується у зовнішню пам'ять журналу тільки при повному заповненні записами.

У зв’язку з тим, що одночасно існують буфер журналу та буфери сторінок оперативної пам'яті, які містять пов'язану інформацію, необхідно виробити деяку загальну політику виштовхування цих буферів, яка б забезпечувала можливість відновлення стану бази даних після збоїв.

Основним принципом узгодженої політики виштовхування буферу журналу та буферів сторінок бази даних є те, що запис про зміну об'єкта БД має потрапляти у зовнішню пам'ять журналу раніше, ніж змінений об'єкт опиниться у зовнішній пам'яті бази даних. Цим вимогам задовольняє протокол WAL – «пиши спочатку у журнал». При цьому, якщо потрібно виштовхнути у зовнішню пам'ять змінений об'єкт бази даних, то перед цим

потрібно гарантувати виштовхування у зовнішню пам'ять журналу запису про його зміну.

Додаткова умова на виштовхування буферів накладається ще одною вимогою, що кожна транзакція, яка успішно завершилася, має бути зафіксованою у зовнішній пам'яті. При цьому необхідно, щоб спочатку було виштовхнуто буфер журналу, а вже за ним масово – буфери сторінок БД, що змінювалися даною транзакцією. Таке рішення потребує істотних накладних витрат при виконанні операції фіксації транзакції.

Мінімальною вимогою, що гарантує можливість відновлення останнього узгодженого стану БД, є виштовхування при фіксації транзакції у зовнішню пам'ять журналу всіх записів про зміну бази даних цією транзакцією. При цьому останнім записом у журналі, який здійснюється від імені даної транзакції, є спеціальний запис про кінець транзакції.

Розглянемо порядок відновлення бази даних у різних ситуаціях з використанням загального для всіх транзакцій журналу і загальною буферизацією записів.

12.2 Відновлення бази даних після збоїв

Спочатку розглянемо порядок відновлення БД під час виконання **індивідуального відкату транзакції.** Для того, щоб можна було виконати індивідуальний відкат транзакції з використанням загального журналу, всі записи у журналі щодо даної транзакції вводяться у зворотний список.

Початком списку для транзакцій, які не закінчилися, є запис про останню зміну бази даних, що була проведена цією транзакцією. Для транзакцій, які закінчилися, початком списку є запис про кінець транзакції, яку буде обов'язково виштовхнуто у зовнішню пам'ять журналу. Необхідно зазначити, що індивідуальні відкати таких транзакцій вже є неможливими.

Кінцем списку завжди є перший запис про зміну бази даних, що була проведена даною транзакцією. Як правило, у кожному записі проставляється унікальний ідентифікатор транзакції, щоб можна було відновити прямий список записів про зміни бази даних цією транзакцією.

Розглянемо індивідуальний відкат транзакцій, що не закінчилися. Спочатку вибирається черговий запис із списку даної транзакції, а потім виконується протилежна за сенсом операція, наприклад, замість операції **INSERT** виконується відповідна операція **DELETE**, замість операції **DELETE** – операція **INSERT** і т.д. Будь-яка з цих зворотних операцій також журналізується. При успішному завершенні відкату у журнал заноситься запис про кінець транзакції, що стає зафіксованою.

Розглянемо порядок **відновлення БД після м'якого збою.** Необхідно зазначити, що одна логічна операція зміни БД може змінювати декілька фізичних блоків БД, наприклад, сторінку даних та декілька сторінок індексів. При цьому сторінки бази даних буферизуються в оперативній пам'яті та виштовхуються незалежно. У разі м'якого збою

набір сторінок зовнішньої пам'яті бази даних може виявитися неузгодженим, тобто частина сторінок зовнішньої пам'яті відповідає об'єкту до зміни, а частина – після зміни. Отже стан зовнішньої пам'яті бази даних після м'якого збою може виявитися фізично не узгодженим.

Стан зовнішньої пам'яті бази даних називається фізично узгодженим, якщо набори сторінок всіх об'єктів є узгодженими, тобто відповідають стану об'єкта або після його зміни, або до зміни.

Розглянемо ситуацію, коли у журналі БД визначено **точки фізичної узгодженості бази даних**. Ці точки являють собою моменти часу, в яких у зовнішній пам'яті містяться узгоджені результати операцій, що завершилися до відповідного моменту часу, і відсутні результати операцій, які не завершилися, а буфер журналу було виштовхнуто у зовнішню пам'ять. Як правило, такі точки (моменти часу) позначають **tpc** (time of physical consistency).

На рисунку 12.1 показано можливі стани транзакцій до моменту м’якого збою. Розглянемо, які дії необхідно виконати з транзакціями за умови, що на момент часу **t1pc** деяким способом вдалося відновити зовнішню пам'ять бази даних.

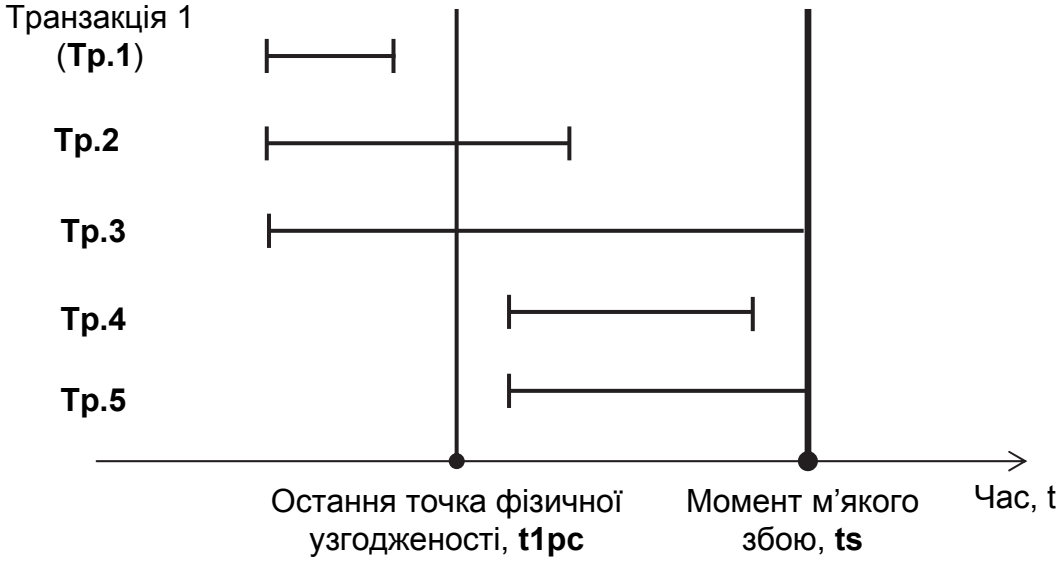


Рисунок 12.1 – Можливі стани транзакцій до моменту м’якого збою

Як видно з цього рисунка, є можливими п’ять станів транзакцій до моменту м’якого збою, які позначені **Тр.1** – **Тр.5**.

Для стану, в якому знаходиться транзакція **Тр.1**, ніяких дій проводити не потрібно, тому що вона закінчилася до моменту **t1pc**, і всі її результати відображені (збережені) у зовнішній пам'яті бази даних.

Для стану, в якому знаходиться транзакція **Тр.2**, потрібно повторно виконати частину операцій, що залишилися (**redo**), тому що у зовнішній пам'яті повністю є відсутніми результати операцій, які виконувалися у транзакції **Тр.2** після моменту останньої фізичної узгодженості БД (**t1pc**).

Отже, повторна пряма інтерпретація операцій **Тр.2** є коректною і приведе до логічно узгодженого стану бази даних. Необхідно зазначити, що у журналі будуть знаходитися записи про всі зміни, які були проведені цією транзакцією, тому що транзакція **Тр.2** успішно завершилася до моменту м'якого збою.

Для стану, в якому знаходиться транзакція **Тр.3**, потрібно виконати у зворотному напрямі першу частину операцій (**undo**), тому що у зовнішній пам'яті бази даних повністю є відсутніми результати операцій **Тр.3**, які були виконані після моменту останньої фізичної узгодженості БД (**t1pc**). Однак у зовнішній пам'яті гарантовано є присутніми результати операцій **Тр.3**, які були виконані до моменту **t1pc**. Тому зворотна інтерпретація операцій **Тр.3** є коректною і приведе до узгодженого стану бази даних. Таким чином, при відновленні БД необхідно усунути всі наслідки виконання транзакції **Тр.3** у зв’язку з тим, що вона не завершилася до моменту м'якого збою.

Для стану, в якому знаходиться транзакція **Тр.4**, яка встигла початися після моменту останньої фізичної узгодженості БД (**t1pc**) і закінчилася до моменту м'якого збою, потрібно виконати повну повторну пряму інтерпретацію операцій (**redo**), тому що у зовнішній пам'яті бази даних повністю є відсутніми результати операцій **Тр.4**.

Для стану, в якому знаходиться транзакція **Тр.5**, яка почалася після моменту останньої фізичної узгодженості БД (**t1pc**) і не встигла завершитися до моменту м'якого збою, ніяких дій виконувати не потрібно, тому що результати операцій цієї транзакції **Тр.5** повністю є відсутніми у зовнішній пам'яті бази даних.

Для фізичної узгодженості бази даних (створення точок фізичної узгодженості бази даних) використовуються два підходи: підхід, що основується на **використанні тіньового механізму**, та підхід, в якому застосовується **журналізація посторінкових змін** бази даних.

Розглянемо фізичну узгодженість БД, що досягається під час застосування першого підходу. При відкритті файла таблиця перетворення номерів його логічних блоків в адреси фізичних блоків зовнішньої пам'яті вводиться в оперативну пам'ять. При модифікації будь-якого блока файла у зовнішній пам'яті виділяється новий блок. При цьому поточна таблиця перетворення (в оперативній пам'яті) змінюється, а тіньова – зберігається незмінною. Якщо під час роботи з відкритим файлом відбувається збій, у зовнішній пам'яті автоматично зберігається стан файла до його відкриття. Для відновлення файла необхідно повторно завантажити в оперативну пам'ять тіньову таблицю перетворення.

Під час використання тіньового механізму періодично виконуються операції встановлення точки фізичної узгодженості бази даних. Для цього всі логічні операції завершуються, всі буфери оперативної пам'яті, вміст яких не відповідає вмісту відповідних сторінок зовнішньої пам'яті, виштовхуються. Тіньова таблиця перетворення файлів бази даних

замінюється на поточну, тобто поточна таблиця перетворення записується замість тіньової.

Перевагою цього підходу є те, що відновлення до точки фізичної узгодженості БД **t1pc** відбувається миттєво, тому що поточна таблиця перетворення просто замінюється на тіньову, а основним недоліком є великі перевитрати зовнішньої пам'яті. У мережі може бути використано вдвічі більше зовнішньої пам'яті, ніж реально потрібно для зберігання бази даних.

Під час застосування іншого підходу фізична узгодженість БД досягається наявністю набору узгоджених сторінок, якому може відповідати свій набір часу. Для цього разом з логічною журналізацією операцій зміни бази даних проводиться журналізація посторінкових змін. При цьому під час відновлення після м'якого збою проводиться посторінковий відкат логічних операцій, що не закінчилися. Останнім записом щодо посторінкових змін від однієї логічної операції є запис про кінець операції. Для того, щоб розпізнати, чи є необхідним відновлення сторінки зовнішньої пам'яті БД, при виштовхуванні будь-якої сторінки з буфера оперативної пам'яті до неї поміщається ідентифікатор останнього запису щодо посторінкової зміни цієї сторінки.

Під час реалізації даного підходу існують два напрями. У першому напрямі здійснюється підтримання загального журналу логічних і сторінкових операцій, що ускладнює структуру журналу, тому що з’являються два види записів, що інтерпретуються абсолютно по-різному. Крім того, істотно збільшується обсяг журналу, тому що зберігаються записи посторінкових змін, які мають локальний характер. Такі недоліки не спостерігаються під час реалізації другого напряму, при якому здійснюється підтримання окремого (короткого) журналу посторінкових змін.

Розглянемо порядок **відновлення БД після жорсткого збою.** Для відновлення останнього узгодженого стану БД після жорсткого збою є недостатньою наявність тільки журналу змін бази даних. У цьому випадку основою відновлення є журнал змін БД та архівна копія бази даних.

Відновлення БД починається зі зворотного копіювання у зовнішню пам'ять бази даних з архівної копії. Надалі для всіх транзакцій, що закінчилися, виконується повна повторна пряма інтерпретація операцій (**redo**). Отже, на основі записів у журналі у прямому напрямі виконуються всі операції транзакцій, що закінчилися, а для транзакцій, які не закінчилися до моменту збою, виконується відкат.

Необхідно зазначити, що до ведення журналу ставляться особливі вимоги щодо надійності його зберігання, але є можливою втрата журналу. У цьому випадку єдиним способом відновлення БД є повернення до архівної копії, але отримати останній узгоджений стан БД уже не вдасться.

Існує декілька способів створення архівних копій БД. Найчастіше архівну копію бази даних створюють у разі переповнення журналу. При

цьому у журнал вводиться так звана «жовта зона», при досягненні якої утворення нових транзакцій тимчасово блокується. Після того як всі транзакції закінчаться, і база даних «прийде» в узгоджений стан, проводиться її архівація, після чого починається нове заповнення журналу. Можна виконувати архівацію бази даних рідше, ніж переповнюється журнал.

У разі переповнення журналу і закінчення всіх початих транзакцій можна архівувати сам журнал. Такий журнал використовують тільки для відтворення архівної копії бази даних, тому журнальну інформацію під час архівації можна стиснути.

Запитання для самоперевірки

1. Яка стратегія застосовується у протоколі ведення журналу WAL ?
2. Які існують варіанти ведення журнальної інформації та у чому їх різниця ?
3. Як журналізація змін БД пов'язана з буферизацією сторінок БД в оперативній пам'яті ?
4. Як проводиться відновлення БД під час індивідуального відкату транзакції ?
5. Як виконується відновлення БД після м'якого збою ?
6. Як створюються точки фізичної узгодженості бази даних ?
7. Як проводиться відновлення БД після жорсткого збою ?

Лекція № 13

ВВЕДЕННЯ У МОВУ БАЗ ДАНИХ SQL

Навчальні цілі:

- розглянути основи мови SQL;
- вивчити порядок використання мови SQL при виборі інформації з таблиць.

Навчальні питання:

1. Основи мови SQL.
2. Використання мови SQL для вибору інформації з таблиць.

13.1 Основи мови SQL

Мова SQL – це структурована мова запитів. Мова SQL дає можливість створювати реляційні бази даних, які являють собою набори пов'язаної інформації, що зберігається у таблицях, також мова SQL дозволяє користувачеві працювати з реляційною БД. Отже, мову SQL орієнтовано спеціально на реляційні бази даних.

Широке застосування різноманітних баз даних привело до необхідності створення стандартної мови, яку можна було б використовувати у різних видах комп'ютерних середовищ. Стандартна мова дає можливість користувачам застосовувати один набір команд для створення, пошуку, зміни та передачі інформації. Крім того, стандартну мову можна однаково використовувати і на персональному комп'ютері, і на мережній робочій станції, і на універсальній ЕОМ.

Стандарт мови SQL визначається за допомогою коду ANSI (Американський національний інститут стандартів) та міжнародної організації зі стандартизації (ISO). Однак код ANSI тільки визначає стандарт мови SQL, а саму мову було створено на фірмі IBM. Інші компанії також брали участь у розвитку мови SQL, наприклад, компанія Oracle має право на ринковий продаж SQL-продуктів. Таким чином, функція ANSI полягає у визначенні стандарту, до якого має бути приведено мову SQL. Однак деякі обмеження стандарту ANSI спонукали розширити корисні функції іншими програмами БД, що не вказані у стандарті ANSI мови SQL. Отже, стандарт ANSI є мінімальною версією мови SQL, і можуть зустрічатися більш розширені версії, але вони мають відповідати вимогам, які визначає цей стандарт.

У стандартній мові SQL є дві версії: інтерактивна та вкладена.

Інтерактивну версію мови SQL використовують для функціонування безпосередньо у базі даних. Вона дозволяє користувачеві самостійно проводити введення й виведення даних. У цій версії мови SQL, якщо команда вводиться користувачем, то вона негайно виконується і

відразу можна побачити результат виконання цієї команди. Інтерактивна версія мови SQL частіше використовується не програмістами, а користувачами БД.

Вкладена версія мови SQL складається з команд мови SQL, які розміщуються всередині програм, які написані, як правило, з використанням іншої мови програмування, що значно розширює функціональні можливості цих програм.

Інтерактивна та вкладена версії мови SQL складаються з декількох частин (підрозділів):

- мови визначення даних (**DDL**), у стандарті ANSI вона називається мовою опису схеми. Ця частина мови SQL складається з команд, за допомогою яких створюються об'єкти (таблиці, індекси, перегляди та ін.) у базі даних;

- мови маніпулювання даними (**DML**) – частини мови SQL, що містить набір команд, які визначають значення, що подані у таблицях у будь-який момент часу;

- мови управління даними (**DCL**), що містить засоби, які визначають можливість дозволити користувачеві виконувати певні дії.

Таким чином, команди мови SQL згруповані за підрозділами стосовно їх функцій.

У полях таблиць реляційної БД можуть знаходитися різні типи значень, які логічно не є однаковими, наприклад, числа і текст. З різними типами даних не можна виконувати деякі операції, наприклад, розміщати числа в алфавітному порядку або вилучати один текст з іншого. У зв’язку з цим різні типи даних мають відрізнятися один від одного так, щоб з ними можна було проводити відповідні процеси та порівняння. У мові SQL це робиться за допомогою призначення кожному полю типу даних, який вказує значення якого типу можуть міститися у цьому полі. Всі значення у даному полі повинні мати однаковий тип.

Стандарт ANSI мови SQL визначає два основних типи: **TEXT** і **NUMBER.** У більшості комерційних програм використовують додаткові спеціальні типи, наприклад, **DATA** (ДАТА) і **TIME** (ЧАС), що є фактично стандартними типами. Деякі БД також підтримують такі типи як, наприклад, **MONEY** (ГРОШІ) та **BINARY** (ДВІЙКОВИЙ). Тип **MONEY** – це спеціальна «валютна» система числення, що використовується комп'ютерами.

Стандарт ANSI мови SQL визначає декілька числових типів, наприклад, **INTEGER** (ЦІЛЕ ЧИСЛО) та **DECIMAL** (ДЕСЯТЕРИЧНЕ ЧИСЛО). Число у типі **INTEGER** можна подати як тип **DECIMAL**, яке не містить цифр праворуч від десяткової крапки.

Текстовим типом у стандарті ANSI мови SQL є тип **CHAR (**СИМВОЛ), який належить до рядка тексту. Поле типу **CHAR** має довжину, яка визначається максимальним числом символів, які можуть бути введені у це поле; як правило, це 254 символи.

Більшість реалізацій БД мають нестандартний текстовий тип **VARCHAR** (ЗМІННЕ ЧИСЛО СИМВОЛІВ). Цей тип є текстовим рядком і може мати будь-яку довжину до визначеного реалізацією максимуму (до 254 символів). Значення типів **CHAR** та **VARCHAR** беруть в одиночні лапки, наприклад, 'текст'.

Відмінність між типами **CHAR** та **VARCHAR** полягає в тому, що для типу **CHAR** резервується кількість пам'яті, що є необхідною для максимальної довжини рядка, а для типу **VARCHAR** пам'ять розподіляється залежно від необхідної довжини рядка і може змінюватися.

Слід зазначити, що текстові типи складаються з усіх друкарських символів, у тому числі й цифр. Однак цифра 1 відрізняється від символу, який позначається ‘1’. Символ являє собою тільки друкарський фрагмент тексту, який не визначається системою як числове значення цифри 1. Наприклад, якщо записана арифметична дія 1 + 1 = 2, то результат буде відповідати цій дії, але якщо записати символи ‘1’ + ‘1’, то результат буде не результатом арифметичної дії, а символом, який задасть користувач (може бути і ‘2’, і ‘4’).

Символьні значення зберігаються у пам’яті комп'ютера як двійкові значення, але показуються користувачеві як друкарський текст. Перетворення виконується системою з використанням одного з двох форматів або ASCII-коду, який застосовується на всіх персональних та малих комп'ютерах, або EBCDIC-коду, який використовується на великих комп'ютерах.

Перед початком роботи з базами даних користувач має з документації з'ясувати, які типи даних підтримує конкретна БД.

Мова SQL, як правило, застосовується у комп'ютерних системах, які мають більше одного користувача, тому необхідно з її допомогою розрізняти їх. Для роботи з такою системою кожен користувач має певний код перевірки доступу, тобто код, що ідентифікує користувача. При цьому сеанс роботи з комп'ютером починається з того, що користувач входить у систему (реєструється), тобто повідомляє про себе. Ідентифікація здійснюється за допомогою певного ID (ідентифікатора). Велика кількість користувачів ID-доступом можуть сприйматися системою як окремі (незалежні) користувачі. Один користувач у різний час може удавати себе за велику кількість користувачів, для цього він застосовує різні ідентифікатори доступу до мови SQL. У більшості середовищ мови SQL дії з БД приведені до спеціального ідентифікатора доступу, який точно відповідає певному користувачеві. При цьому таблиця або інший об'єкт БД належить користувачеві, що його створив. Користувач може мати або не мати привілею на виконання дії з об'єктом БД.

Спеціальне значення **USER** (КОРИСТУВАЧ) можна застосовувати як аргумент у команді, який вказує на доступний ідентифікатор користувача, що видав команду.

У мові SQL є певні умовні позначення, своя термінологія та поняття. Коротко розглянемо їх.

Перше поняття – це **ключові слова**, які застосовуються у мові SQL для виконання певних дій. Ключові слова – це слова, які мають спеціальне значення у мові SQL. Вони можуть бути командами, але не текстом, а також не іменами об'єктів БД. Ключові слова виділяються у мові SQL написанням ВЕЛИКИМИ БУКВАМИ. Ще одним поняттям є **об'єкти,** що являють собою структури БД, яким надано імена та які зберігаються у пам'яті. До них належать базові таблиці, подання та індекси.

Спеціальні терміни у мові SQL використовують для її опису. Серед основних термінів, які є найважливішими для опису та розуміння мови, застосовують такі слова, як “**запит”**, “**речення”** та “**предикат”**. Але безпосередньо у мові SQL ці терміни не означають самостійних дій і не беруть участі у роботі з БД.

Команди мови SQL являють собою інструкції, за допомогою яких звертаються мовою SQL до БД. Команди складаються з однієї або більше окремих логічних частин, які називаються **реченнями**, що містять ключові слова та аргументи, наприклад, **WHERE city = ‘Kharkiv’**. Речення починаються ключовим словом, наприклад, **WHERE**, а аргументи завершують або змінюють значення речення, наприклад, **city = ‘Kharkiv’**.

Основними умовними позначеннями є:

- **квадратні дужки** [ ] – вказують частини, які можуть не використовуватися;
- **багатокрапки ...** – зазначають, що весь попередній текст може повторюватися будь-яку кількість разів;
- **кутові дужки < >** – вказують, що слова у дужках пояснюють дії користувачів, але самостійно ніяких дій не виконують.

13.2 Використання мови SQL для вибору інформації з таблиць

Найчастіше під час роботи з БД користувач виконує операцію запиту інформації з БД. **Запит** – це команда, яка передається програмі бази даних і повідомляє БД, що потрібно вивести певну інформацію з таблиць у пам'ять. Ця інформація, як правило, відображається безпосередньо на екрані комп'ютера. Крім того, інформація з БД може виводитися на принтер, може пересилатися та зберігатися в окремому файлі у пам’яті комп'ютера, а також використовуватися як вхідна інформація для іншої команди або процесу.

За допомогою мови SQL будь-який запит виконується однією командою, яка називається **SELECT** (ВИБРАТИ). Коротко розглянемо порядок застосування цієї команди.

Команда **SELECT** подає інструкцію БД для отримання інформації з таблиці, тобто за допомогою цієї команди виводяться дані з таблиці. У більшості комерційних програм також виводяться заголовки стовпців, а в

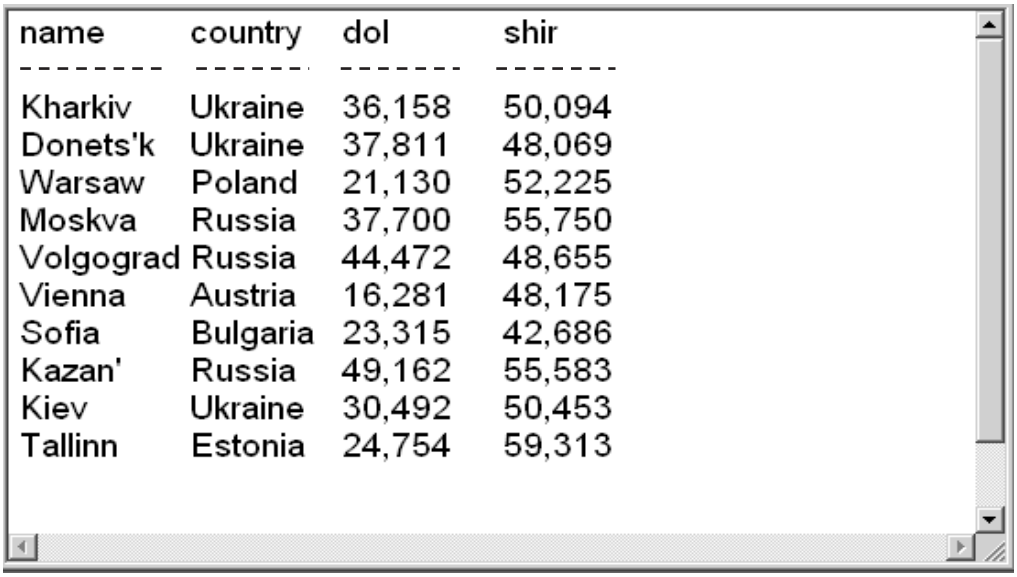
деяких визначається детальне форматування виводу. У прикладах, що наведено нижче, вкажемо заголовки стовпців. Розглянемо приклад виведення даних з таблиці **town**, яку було розглянуто в лекції № 8. Для цього необхідно набрати на клавіатурі комп'ютера таке:

```
SELECT name, country, dol, shir
FROM town;
```

Запит можна складати у декілька рядків, як показано вище, а можна – в один рядок:

```
SELECT name, country, dol, shir FROM town;
```

У результаті виконання команди **SELECT** буде отримано список міст, країн, їх довготу і широту, як показано на рисунку 13.1.



name	country	dol	shir
Kharkiv	Ukraine	36,158	50,094
Donets'k	Ukraine	37,811	48,069
Warsaw	Poland	21,130	52,225
Moskva	Russia	37,700	55,750
Volgograd	Russia	44,472	48,655
Vienna	Austria	16,281	48,175
Sofia	Bulgaria	23,315	42,686
Kazan'	Russia	49,162	55,583
Kiev	Ukraine	30,492	50,453
Tallinn	Estonia	24,754	59,313

Рисунок 13.1 – Результат виконання команди **SELECT** із вибору заданих стовпців таблиці **town**

Розглянемо, що означають різні слова у цьому запиті.

SELECT є ключовим словом (обов'язково пишеться великими буквами), яке повідомляє базу даних, що ця команда – запит. Усі запити починаються цим словом з подальшим пропуском.

Далі наведено назви стовпців із таблиці, які вибираються запитом – **name, country, dol, shir.** Непозначені стовпці не будуть введені у результат виконання (виведення) команди.

FROM також є ключовим словом, що є подібним слову **SELECT**. Ключове слово **FROM** має бути наведено у кожному запиті. Воно супроводжується пропуском та ім'ям таблиці, яка використовується як джерело інформації, наприклад, таблиця **town**.

Крапка з комою (;) використовується у всіх інтерактивних командах мови SQL, щоб повідомити БД, що команда записана та є готовою до

виконання. У деяких системах індикатором кінця команди є **зворотний слеш (\ )** у рядку.

Необхідно зазначити, що запит, який було розглянуто у прикладі, буде виводити невпорядковані дані. Для їх впорядкування використовуються спеціальні команди мови SQL.

Після набирання команди на клавіатурі для її виконання необхідно натиснути клавішу **ENTER**. Для виведення повного списку стовпців таблиці, тобто всієї інформації з таблиці, можна застосовувати **зірочку (*)**, як показано нижче.

```
SELECT *
FROM town;
```

У результаті виконання команди **SELECT** буде отримано повний список стовпців таблиці **town**, як показано на рисунку 13.2.

name	country	population	dol	shir	capital	pop_urb
Kharkiv	Ukraine	1 940 000	36,158	50,094	N	2384,6
Donets'k	Ukraine	2 200 000	37,811	48,069	N	4568
Warsaw	Poland	2 323 000	21,130	52,225	Y	3703,5
Moskva	Russia	13 100 000	37,700	55,750	Y	NULL
Volgograd	Russia	1 360 000	44,472	48,655	N	NULL
Vienna	Austria	1 875 000	16,281	48,175	Y	NULL
Sofia	Bulgaria	1 205 000	23,315	42,686	Y	NULL
Kazan'	Russia	1 140 000	49,162	55,583	N	NULL
Kiev	Ukraine	2 900 000	30,492	50,453	Y	NULL
Tallinn	Estonia	482 000	24,754	59,313	Y	NULL

Рисунок 13.2 – Результат виконання команди **SELECT** із вибору повного списку стовпців таблиці **town**

Із рисунка 13.2 видно, що у стовпці **POP_URB** у деяких рядках записано значення **NULL**. У наступних прикладах це значення також буде застосовано.

Як вже було показано вище, можна здійснювати переглядання тільки певних стовпців таблиці, тобто тільки необхідної інформації з таблиці. Для цього командою **SELECT** просто не виводяться стовпці, які не потрібно переглядати.

За допомогою команди **SELECT** можна також здійснювати упорядковування стовпців. Якщо використовувалася у запиті зірочка (*), то всі стовпці будуть показані у тому порядку, в якому вони зберігаються у пам’яті комп’ютера. Однак, якщо стовпці таблиці є впорядкованими, це не означає, що вони будуть виводитися у необхідному порядку. Для

виведення стовпців таблиці у необхідному порядку потрібно вказати командою **SELECT** стовпці у цьому порядку.

Під час роботи з БД також може виникнути необхідність виведення даних, що не дублюються (не повторюються), тобто видалення надмірних даних. Для цього використовується команда **DISTINCT** (ВІДМІННІСТЬ), яка забезпечує усунення дублюючих значень з команди **SELECT**, наприклад, якщо необхідно визначити, які країни знаходяться у даний час у таблиці **town**. Для цього необхідно набрати на клавіатурі комп'ютера таке:

```
SELECT country
FROM town;
```

У результаті виконання команди **SELECT** буде отримано весь список країн з дублюванням, як показано на рисунку 13.3.

Рисунок 13.3 – Результат виконання команди **SELECT** із дублюванням назв країн

Для отримання списку без дублікатів необхідно набрати на клавіатурі комп'ютера таке:

```
SELECT DISTINCT country
FROM town;
```

У результаті виконання команди **SELECT** у цьому випадку буде отримано список країн без дублювання, як показано на рисунку 13.4.

Команду **DISTINCT** можна вказувати тільки один раз в одному конкретному реченні. Якщо речення вибирає декілька полів, команда **DISTINCT** пропускає рядки, де всі вибрані поля є однаковими. Рядки, в яких деякі значення однакові, а деякі – різні, не пропускаються і будуть збережені. Для скасування дій, виконаних за командою **DISTINCT**, тобто для відновлення дублювання рядків, необхідно замість команди **DISTINCT** задати команду **ALL.**

Рисунок 13.4 – Результат виконання команди **SELECT** без дублювання назв країн

Під час роботи з таблицями великих розмірів користувачеві, як правило, необхідно мати конкретну інформацію, тобто тільки окремі рядки. За допомогою мови SQL можна встановлювати критерії вибору рядків з таблиці для виведення на екран. Для цього використовується команда **WHERE**, яка дозволяє встановлювати предикати, умови яких можуть бути або правильними (true), або неправильними (false) для будь-якого рядка таблиці. Команда **WHERE** дозволяє вибрати тільки ті рядки з таблиці, для яких таке твердження є правильним.

Наприклад, якщо необхідно вибрати з таблиці **town** назви міст, які розташовані на території України, то для цього слід набрати на клавіатурі комп'ютера таке:

```
SELECT name, country
FROM town
WHERE country = 'Ukraine';
```

При цьому під час роботи програми бази даних переглядається вся таблиця рядок за рядком та досліджується кожен рядок, щоб визначити, відповідає він заданій умові або не відповідає.

У результаті виконання команди **SELECT** буде отримано список міст, які розташовані на території України, як показано на рисунку 13.5.

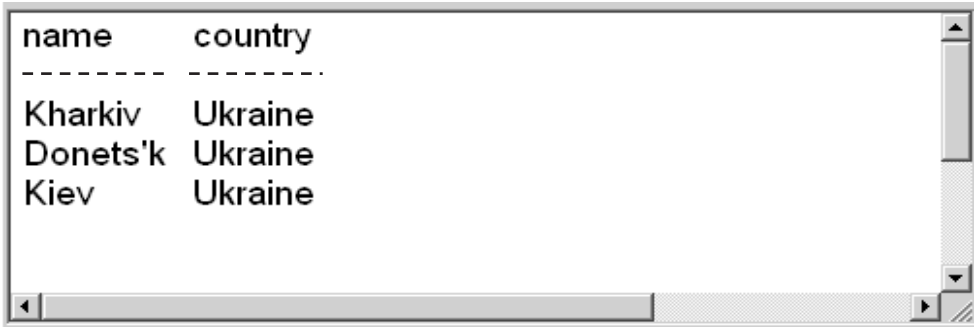


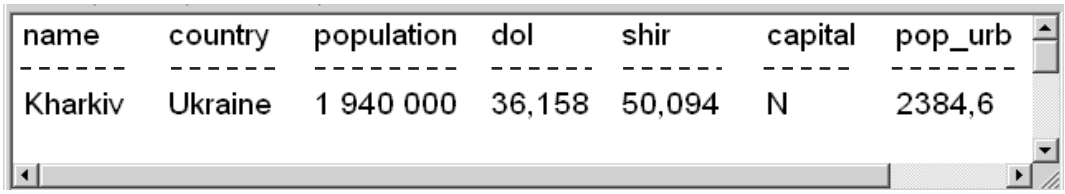
Рисунок 13.5 – Результат виконання команди **SELECT** із вибору міст, які розташовані на території України

Наприклад, якщо необхідно вибрати з таблиці **town** міста, населення в яких дорівнює 1 940 000 людей (числове поле), то для цього необхідно набрати на клавіатурі комп'ютера таке:

```
SELECT *
FROM town
WHERE population = 1 940 000;
```

При цьому одиночні лапки не використовують, тому що умова – це числове поле.

У результаті виконання команди **SELECT** буде отримано список міст з населенням 1 940 000 людей, як показано на рисунку 13.6.



name	country	population	dol	shir	capital	pop_urb
Kharkiv	Ukraine	1 940 000	36,158	50,094	N	2384,6

Рисунок 13.6 – Результат виконання команди **SELECT** із вибору міст з населенням 1 940 000 людей

У мові SQL також використовують основні булеві операції, що поєднують правильні/неправильні значення та створюють одне правильне або одне неправильне значення. Стандартними булевими операціями, які реалізовані у мові SQL, є **AND, OR** та **NOT**.

У результаті виконання операції **AND** вибирають два булева значення як аргументи (у формі ***A* AND *B***) та перевіряють чи правильними є обидва аргументи. У результаті виконання операції **OR** вибирають два булева значення як аргументи (у формі ***A* OR *B***) та проводять оцінювання: чи правильним є один з аргументів. У результаті виконання операції **NOT** вибирають одиночне булеве значення як аргумент (у формі **NOT *A***) та замінюють його з неправильного на правильне або з правильного на неправильне.

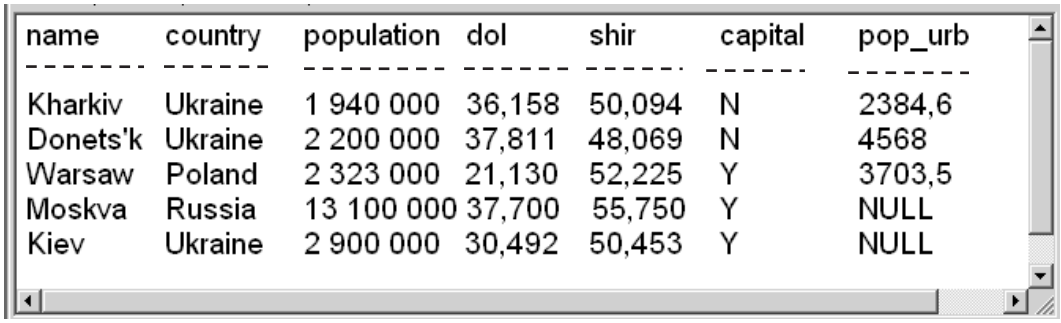
Необхідно зазначити, що операцію **NOT** слід застосовувати перед булевою операцією, значення якої має змінитися і яке не застосовують безпосередньо перед реляційною операцією. Наприклад, неправильне задавання предиката оцінювання буде таким:

```
population NOT > 2 000 000
```

За допомогою мови SQL можна застосовувати операцію **NOT** з булевим виразом, який розміщується після нього. Для цього необхідно набрати на клавіатурі комп'ютера таке:

```
SELECT *
FROM town
WHERE NOT (country = 'Ukraine' OR population > 2 000 000);
```

У результаті виконання команди **SELECT** буде отримано список міст України або міст з населенням більше 2 000 000 людей, як показано на рисунку 13.7.



name	country	population	dol	shir	capital	pop_urb
Kharkiv	Ukraine	1 940 000	36,158	50,094	N	2384,6
Donets'k	Ukraine	2 200 000	37,811	48,069	N	4568
Warsaw	Poland	2 323 000	21,130	52,225	Y	3703,5
Moskva	Russia	13 100 000	37,700	55,750	Y	NULL
Kiev	Ukraine	2 900 000	30,492	50,453	Y	NULL

Рисунок 13.7 – Результат виконання команди **SELECT** із вибору міст України або міст з населенням більше 2 000 000 людей

Круглі дужки у мові SQL означають, що все, що знаходиться всередині них обчислюється у першу чергу та обробляється як єдиний вираз. Отже, у прикладі, що розглядається, береться кожний рядок і визначається, чи відповідає істині рівність **country = 'Ukraine'** або нерівність **population > 2 000 000**. Якщо будь-яка умова правильна, булів вираз всередині круглих дужок також є правильним. Однак, якщо булів вираз всередині круглих дужок є правильним, а предикат усього цілого буде неправильним, то операція **NOT** перетворить правильно у неправильно та навпаки.

Існують інші, складніші булеві операції (типу "**виключне АБО**"), але вони можуть бути сформовані з цих трьох простих операцій. Пов'язуючи предикати з булевими операціями, можна значно розширяти їх можливості. Булеві операції під час їх окремого застосування є простими, але вони не є такими простими, коли їх комбінують у комплексний вираз. Для правильного застосування комплексних виразів булевих операцій доцільно спочатку оцінювати булеві вирази, що знаходяться у перших круглих дужках, далі об'єднувати їх в єдине булеве значення, а потім об'єднувати його з розміщеними далі значеннями і т.д.

Запитання для самоперевірки

1. Як характеризується мова SQL, що визначається стандартом ANSI ?
2. У чому полягає різниця між інтерактивною і вкладеною мовами SQL ?
3. З яких частин (підрозділів) складається мова SQL ?
4. З якою метою у мові SQL застосовують типи ?
5. У чому полягає різниця між ключовими словами і термінами ?
6. Які існують особливості застосування запитів за допомогою команди **SELECT** ?
7. Що визначає і як застосовується команда **WHERE** у запитах ?

Лекція № 14

СПЕЦІАЛЬНІ ОПЕРАТОРИ МОВИ SQL

Навчальні цілі:
- вивчити порядок використання спеціальних операторів у мові SQL;
- розглянути види агрегатних функцій мови SQL.

Навчальні питання:
1. Використання спеціальних операторів.
2. Агрегатні функції.

14.1 Використання спеціальних операторів

У мові SQL використовується декілька спеціальних операторів, які застосовуються для визначення умов у запитах. До основних з них належать оператори: **IN; BETWEEN; LIKE; IS NULL.** Коротко розглянемо порядок їх застосування.

Оператор **IN** визначає набір значень, до складу яких певне значення або може входити, або не може. Наприклад, необхідно вибрати (знайти) міста, які знаходяться або в Україні або у Болгарії. Для цього необхідно набрати на клавіатурі комп'ютера таке:

```
SELECT *
FROM town
WHERE country = 'Ukraine' OR country = 'Bulgaria';
```

У результаті виконання команди **SELECT** буде отримано список міст України та Болгарії, як показано на рисунку 14.1.

name	country	population	dol	shir	capital	pop_urb
Kharkiv	Ukraine	1 940 000	36,158	50,094	N	2384,6
Donets'k	Ukraine	2 200 000	37,811	48,069	N	4568
Sofiya	Bulgaria	1 205 000	23,315	42,686	Y	NULL
Kiev	Ukraine	2 900 000	30,492	50,453	Y	NULL

Рисунок 14.1 – Результат виконання команди **SELECT** із вибору міст України та Болгарії

Можна також застосувати більш простий спосіб отримання тієї ж самої інформації. Для цього в операторі **IN** визначається у круглих дужках та розділених комами набір імен значень вибору. При цьому необхідно набрати на клавіатурі комп'ютера таке:

```
SELECT *
FROM town
WHERE country IN ('Ukraine', 'Bulgaria');
```

Під час виконання операції оператор **IN** перевіряє різні значення вказаного поля та намагається знайти їх збіг зі значеннями з набору. Якщо збіг відбувається, то предикат є правильним і значення виводяться. Якщо набір містить числові значення, одиночні лапки вилучаються.

Оператор **BETWEEN** є схожим на оператор **IN**, але він визначає діапазон, в якому мають знаходитися вибрані значення. При цьому значення мають зменшуватися, що робить предикат правильним. Спочатку вводиться ключове слово **BETWEEN** з початковим значенням, потім ключове **AND** і наприкінці кінцеве значення. На відміну від оператора **IN** оператор **BETWEEN** є чутливим до порядку значень. Перше значення у реченні має бути останнім за алфавітним або числовим порядком.

Наприклад, якщо необхідно вибрати з таблиці **town** міста, населення в яких дорівнює від 1 940 000 до 1 140 000 людей, то для цього необхідно набрати на клавіатурі комп'ютера таке:

```
SELECT *
FROM town
WHERE population BETWEEN 1 940 000 AND 1 140 000;
```

У результаті виконання команди **SELECT** буде отримано список міст з населенням від 1 940 000 до 1 140 000 людей, як показано на рисунку 14.2.

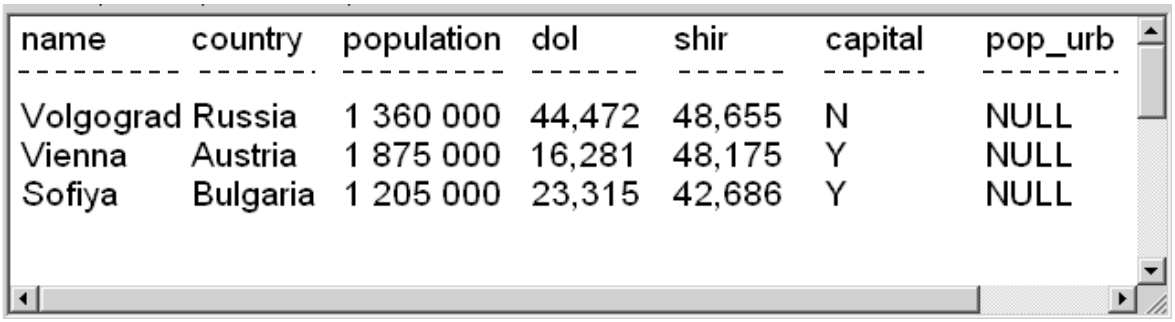
name	country	population	dol	shir	capital	pop_urb
Kharkiv	Ukraine	1 940 000	36,158	50,094	N	2384,6
Volgograd	Russia	1 360 000	44,472	48,655	N	NULL
Vienna	Austria	1 875 000	16,281	48,175	Y	NULL
Sofiya	Bulgaria	1 205 000	23,315	42,686	Y	NULL
Kazan'	Russia	1 140 000	49,162	55,583	N	NULL

Рисунок 14.2 – Результат виконання команди **SELECT** із вибору міст з населенням від 1 940 000 до 1 140 000 людей

У прикладі (див. рисунок 14.2) оператор **BETWEEN** виводить граничні значення, що позначені у запиті, однак, щоб їх не виводити, необхідно набрати на клавіатурі комп'ютера таке:

```
SELECT *
FROM town
WHERE (population BETWEEN 1 940 000 AND 1 140 000)
AND NOT population IN (1 940 000, 1 140 000);
```

У результаті виконання команди **SELECT** буде отримано список міст з населенням від 1 940 000 до 1 140 000 людей, але міста з цими граничними значеннями виводитися не будуть, як показано на рисунку 14.3.



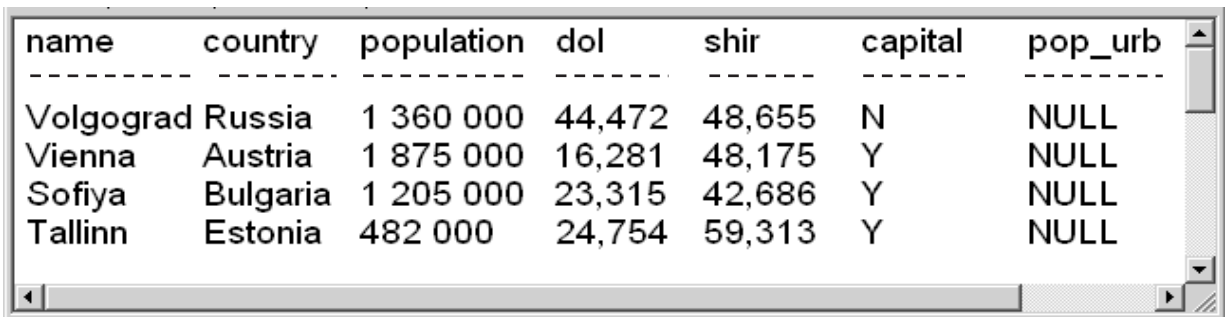
name	country	population	dol	shir	capital	pop_urb
Volgograd	Russia	1 360 000	44,472	48,655	N	NULL
Vienna	Austria	1 875 000	16,281	48,175	Y	NULL
Sofiya	Bulgaria	1 205 000	23,315	42,686	Y	NULL

Рисунок 14.3 – Результат виконання команди **SELECT** із вибору міст з населенням від 1 940 000 до 1 140 000 людей за винятком граничних значень

Оператор **BETWEEN** також може працювати з символьними полями у термінах еквівалентів ASCII, тобто можна використовувати оператор **BETWEEN** для вибору ряду символьних значень з упорядкованих за алфавітом значень. Наприклад, необхідно вибрати з таблиці **town** всі міста, назви яких потрапили в алфавітний діапазон від літери **S** до літери **V**. Для цього необхідно набрати на клавіатурі комп'ютера таке:

```
SELECT *
FROM town
WHERE name BETWEEN 'V' AND 'S';
```

У результаті виконання команди **SELECT** буде отримано список міст, назви яких потрапили в алфавітний діапазон від літери **S** до літери **V**, як показано на рисунку 14.4.



name	country	population	dol	shir	capital	pop_urb
Volgograd	Russia	1 360 000	44,472	48,655	N	NULL
Vienna	Austria	1 875 000	16,281	48,175	Y	NULL
Sofiya	Bulgaria	1 205 000	23,315	42,686	Y	NULL
Tallinn	Estonia	482 000	24,754	59,313	Y	NULL

Рисунок 14.4 – Результат виконання команди **SELECT** із вибору міст, назви яких потрапили у заданий алфавітний діапазон

Оператор **LIKE** застосовується тільки з полями типу **CHAR** або **VARCHAR** для пошуку необхідних рядків.

З оператором **LIKE** використовуються два типи шаблонів:

- **символ підкреслення ‘_’** заміщає будь-який одиночний символ у слові. Наприклад, 'b_t' відповідатиме словам 'bat' або 'bit', але не відповідатиме 'brat';

- **знак відсотка (%)** заміщає послідовність будь-якої кількості символів у слові (включаючи символи нуля). Наприклад, '%p%t' відповідатиме словам 'put', 'posit', або 'opt', але не 'spite'.

Наприклад, якщо необхідно вибрати з таблиці **town** всі міста, назви яких починаються з літери **K**, то необхідно набрати на клавіатурі комп’ютера таке:

```
SELECT *
FROM town
WHERE name LIKE 'K%';
```

У результаті виконання команди **SELECT** буде отримано список міст, назви яких починаються з літери **K**, як показано на рисунку 14.5.

name	country	population	dol	shir	capital	pop_urb
Kharkiv	Ukraine	1 940 000	36,158	50,094	N	2384,6
Kazan'	Russia	1 140 000	49,162	55,583	N	NULL
Kiev	Ukraine	2 900 000	30,492	50,453	Y	NULL

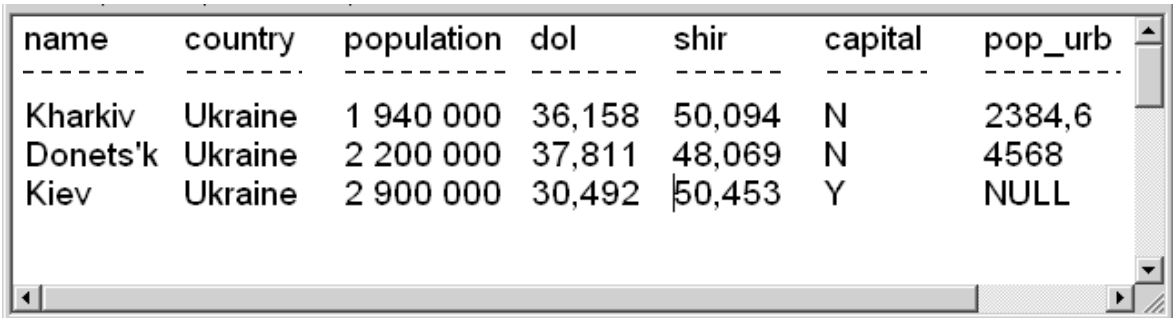
Рисунок 14.5 – Результат виконання команди **SELECT** із вибору міст, назви яких починаються з літери **K**

Оператор **LIKE** може бути зручним, якщо необхідно знайти ім'я або інше значення, які невідомо, як точно записувати. Наприклад, необхідно вибрати з таблиці **town** назви міст, які розташовано на території України, але користувачеві невідомо, як слово Україна писати англійською мовою. Для цього необхідно набрати на клавіатурі комп’ютера таке:

```
SELECT *
FROM town
WHERE country 'U____ne %';
```

У результаті виконання команди **SELECT** буде отримано список міст України, як показано на рисунку 14.6.

Груповий символ **'%'** у кінці рядка є необхідним, якщо довжина поля **name** є більшою, ніж число символів у слові **Ukraine**. Груповий символ **'%'** просто відповідає пропускам. Якщо поле **name** має тип **VARCHAR**, то груповий символ **'%'** можна не використовувати.



name	country	population	dol	shir	capital	pop_urb
Kharkiv	Ukraine	1 940 000	36,158	50,094	N	2384,6
Donets'k	Ukraine	2 200 000	37,811	48,069	N	4568
Kiev	Ukraine	2 900 000	30,492	50,453	Y	NULL

Рисунок 14.6 – Результат виконання команди **SELECT** із вибору міст України з використанням оператора **LIKE**

У разі, якщо потрібно знайти символ відсотка (**%**) або символ підкреслення (**_**) у рядку, то у предикаті оператора **LIKE** необхідно визначити будь-який одиночний символ як символ **ESC** (коса лінія – **/**). Символ **ESC** використовується у предикаті відразу перед відсотком або підкресленням та означає, що відсоток або підкреслення є символом, який необхідно знайти. Наприклад, слід знайти у таблиці **town** значення **name**, де є підкреслення. Для цього необхідно набрати на клавіатурі комп'ютера таке:

```
SELECT *
FROM town
WHERE name LIKE '%/_';
```

Необхідно зазначити, що символ **ESC** має бути одиночним і його слід застосовувати тільки відразу після одиночного символу. Крім того, символ **ESC** можна також використовувати самостійно, тобто, якщо необхідно знайти стовпець із символом **ESC**, то його слід вводити два рази один за одним. Перший символ **ESC** визначає одиночний символ, а другий символ **ESC** є самостійним символом, що необхідно знайти.

Під час робот з таблицями БД можуть з'явитися записи, які не мають ніяких значень поля. Така ситуація може виникнути, наприклад, тому що інформація не завершена, або тому що це поле просто не заповнювалося. Мова SQL дозволяє враховувати таку ситуацію шляхом застосування (введення) значення **NULL** (ПОРОЖНІЙ) у полі замість будь-якого значення. Якщо поле позначено **NULL**, то програма бази даних спеціально промаркірувала це поле як те, що не містить ніякого значення для цього рядка (запису).

Призначення полю значення **NULL** відрізняється від розміщення у полі значення 0 (нуль) або пропуску. Значення 0 (нуль) або пропуск у базі даних обробляються так само, як і будь-яке інше значення. У той же час **NULL** не є технічним значенням, воно не має типу даних та може розміщатися у полі будь-якого типу.

Оператор **IS NULL** застосовується із значенням **NULL** у разі, якщо **NULL** порівнюється з будь-яким значенням, навіть з іншим таким же значенням **NULL**. У разі такого порівняння результат буде ні **true**, ні **false**, він буде невідомим (**undefined**). Отже, результат виразу типу **'city = NULL'** або виразу типу **'city IN (NULL)'** буде невідомим у будь-якому випадку. Для того, щоб розрізняти неправильно (**false**) та невідомо (**undefined**), використовується спеціальний оператор **IS**, який застосовується разом із ключовим словом **NULL**, для розміщення значення **NULL**. Наприклад, необхідно знайти всі записи у таблиці **town** із значеннями **NULL** у стовпці **pop_urb**. Для цього необхідно набрати на клавіатурі комп'ютера таке:

```
SELECT *
FROM town
WHERE pop_urb IS NULL;
```

У результаті виконання команди **SELECT** буде отримано список міст зі значеннями **NULL** у стовпці **pop_urb**, як показано на рисунку 14.7.

name	country	population	dol	shir	capital	pop_urb
Moskva	Russia	13 100 000	37,700	55,750	Y	NULL
Volgograd	Russia	1 360 000	44,472	48,655	N	NULL
Vienna	Austria	1 875 000	16,281	48,175	Y	NULL
Sofiya	Bulgaria	1 205 000	23,315	42,686	Y	NULL
Kazan'	Russia	1 140 000	49,162	55,583	N	NULL
Kiev	Ukraine	2 900 000	30,492	50,453	Y	NULL
Tallinn	Estonia	482 000	24,754	59,313	Y	NULL

Рисунок 14.7 – Результат виконання команди **SELECT** із вибору міст із значеннями **NULL** у стовпці **pop_urb**

Під час робот з таблицями БД часто використовують булевий оператор **NOT** зі спеціальними операторами. Його застосовують, коли необхідно виконати протилежні операції, що вводяться. У цьому випадку оператор **NOT** має знаходитися перед виразом, що вводиться.

Наприклад, якщо необхідно усунути **NULL** з виведення, використовують оператор **NOT**, щоб змінити значення предиката на протилежне . Для цього необхідно набрати на клавіатурі комп'ютера таке:

```
SELECT *
FROM town
WHERE pop_urb NOT IS NULL;
```

Можна застосувати інший спосіб для отримання тієї ж самої інформації. При цьому необхідно набрати на клавіатурі комп'ютера таке:

```
SELECT *
FROM town
WHERE pop_urb97 IS NOT NULL;
```

У результаті виконання команди **SELECT** буде отримано список міст, для яких у стовпці **pop_urb** не має значень **NULL**, як показано на рисунку 14.8.

name	country	population	dol	shir	capital	pop_urb
Kharkiv	Ukraine	1 940 000	36,158	50,094	N	2384,6
Donets'k	Ukraine	2 200 000	37,811	48,069	N	4568
Warsaw	Poland	2 323 000	21,130	52,225	Y	3703,5

Рисунок 14.8 – Результат виконання команди **SELECT** із вибору міст, для яких у стовпці **pop_urb** є відсутніми значення **NULL**

Можна використовувати булевий оператор **NOT** з оператором **IN**. Наприклад, необхідно вибрати міста, яких немає ні в Україні, ні у Болгарії. Для цього необхідно набрати на клавіатурі комп'ютера таке:

```
SELECT *
FROM town
WHERE NOT country IN ('Ukraine', 'Bulgaria');
```

У результаті виконання команди **SELECT** буде отримано список міст, які є у таблиці **town**, за виключенням міст, які заходяться в Україні та Болгарії, як показано на рисунку 14.9.

name	country	population	dol	shir	capital	pop_urb
Warsaw	Poland	2 323 000	21,130	52,225	Y	3703,5
Moskva	Russia	13 100 000	37,700	55,750	Y	NULL
Volgograd	Russia	1 360 000	44,472	48,655	N	NULL
Vienna	Austria	1 875 000	16,281	48,175	Y	NULL
Kazan'	Russia	1 140 000	49,162	55,583	N	NULL
Tallinn	Estonia	482 000	24,754	59,313	Y	NULL

Рисунок 14.9 – Результат виконання команди **SELECT** із вибору міст, за виключенням тих, які знаходяться в України та Болгарії

Таким же способом можна використовувати булевий оператор **NOT** з операторами **BETWEEN** та **LIKE** (**NOT BETWEEN** та **NOT LIKE**).

14.2 Агрегатні функції

Під час роботи з БД користувач може отримувати інформацію із застосуванням так званих агрегатних (узагальнювальних) функцій. Такі функції вибирають з групи значень БД із поля та перетворюють їх у одиночне значення.

Основними агрегатними функціями, що застосовуються у мові SQL, є:

- **COUNT**, яка визначає кількість рядків або не **NULL**-значень полів, які вибрав запит;
- **SUM**, яка розраховує арифметичну суму всіх вибраних значень даного поля;
- **AVG**, яка проводить усереднення всіх вибраних значень цього поля;
- **MAX**, яка знаходить найбільше з усіх вибраних значень цього поля;
- **MIN**, яка знаходить найменше з усіх вибраних значень даного поля.

Коротко розглянемо порядок застосування агрегатних функцій під час роботи з БД.

Особливістю застосування агрегатних функцій є те, що вони вибирають імена полів як аргументи та виконують з ними дії. Функції **SUM** та **AVG** використовують тільки з числовими полями, а до полів з символами ці функції не застосовують. З функціями **COUNT**, **MAX** та **MIN** можна використовувати і числові, і символьні поля. Під час роботи функції **MAX** та **MIN** перетворюють символьні поля в еквівалент ASCII. Після цього функції виконують пошук мінімального (першого) (функція **MIN**) або максимального (останнього) (функція **MAX**) значення в алфавітному порядку.

Наприклад, необхідно визначити загальну кількість населення всіх міст, що розміщенні у таблиці **town**. Для цього необхідно набрати на клавіатурі комп'ютера таке:

```
SELECT SUM (population)
FROM town;
```

Результатом виконання даної функції буде число **28 525 000**, яке є кількістю (сумою) населення всіх міст таблиці **town.**

Аналогічно можна знайти усереднену кількість населення всіх міст, що розміщенні у таблиці **town**. Для цього необхідно набрати на клавіатурі комп'ютера таке:

```
SELECT AVG (population)
FROM town;
```

Результатом виконання даної функції буде число **2 852 500**, яке є усередненою кількістю (сумою) населення всіх міст таблиці **town.**

Із застосуванням функцій **MAX** та **MIN** можна визначити найбільшу та найменшу кількість населення у містах, що розміщенні у таблиці **town**. Для цього необхідно набрати на клавіатурі комп'ютера таке:

```
SELECT MAX (population), MIN (population)
FROM town;
```

Результатом виконання функції буде два числа **13 100 000** та **482 000**, які відповідають максимальній та мінімальній кількості населення у містах у таблиці **town.**

Функція **COUNT** визначає число значень у даному стовпці або число рядків у таблиці. У разі визначення кількості значень у стовпці функція **COUNT** використовується разом з командою **DISTINCT**, що дозволяє визначити число різних значень у стовпці.

Наприклад, якщо необхідно визначити кількість країн (**country**), описаних у таблиці **town**, то необхідно набрати на клавіатурі комп'ютера таке:

```
SELECT COUNT (DISTINCT country)
FROM town;
```

Результатом виконання даної функції буде число **6**, яке відповідає кількості країн у таблиці **town.**

Необхідно зазначити, що команда **DISTINCT** супроводжується ім'ям поля, з яким вона застосовується (**country**), і разом з ним розміщується у круглі дужки, але не відразу після команд **SELECT**. Команду **DISTINCT** можна використовувати з будь-якою агрегатною функцією, але найчастіше її застосовують з функцією **COUNT**. Використання команди **DISTINCT** з функціями **MAX** та **MIN є** недоцільним, тому що ці функції вибирають тільки одне значення, а під час застосування **SUM** та **AVG** ці функції працюють з усіма повторюваними значеннями.

У разі необхідності визначення загальної кількості рядків у таблиці використовують функцію **COUNT** із зірочкою замість імені поля. Для цього необхідно набрати на клавіатурі комп'ютера таке:

```
SELECT COUNT (*)
FROM town;
```

Результатом виконання даної функції буде число **10**, яке відповідає кількості рядків у таблиці **town.** Необхідно зазначити, що функція **COUNT** із зірочкою містить і значення **NULL**, і дублікати.

Агрегатні функції можна також використовувати з командою **ALL,** яка поміщається перед ім'ям поля подібно команді **DISTINCT**, але вона означає протилежне – включати дублікати.

Між використанням у функції **COUNT** команди **ALL** та зірочки (*****) існують відмінності. Так, команда **ALL** застосовує ім'я поля як аргумент і не може підрахувати значення **NULL**. У той же час використання зірочки (*****) дозволяє ввести значення **NULL** у розрахунок, однак її застосовують з функцією **COUNT**.

Наприклад, для визначення кількості не **NULL**-значень у полі **pop_urb** таблиці **town** за допомогою функції **COUNT**, включаючи повторення, необхідно набрати на клавіатурі комп'ютера таке:

```
SELECT COUNT (ALL pop_urb)
FROM town;
```

Результатом виконання даної функції буде число **3**, яке відповідає кількості не **NULL**-значень у полі **pop_urb** таблиці **town.**

Запитання для самоперевірки

1. У чому полягає різниця між операторами **IN** та **BETWEEN** ?
2. Для яких полів та з якою метою застосовується оператор **LIKE** ?
3. Що потрібно ввести в оператор **LIKE**, якщо необхідно знайти символ відсотка (%) ?
4. Для чого застосовують значення **NULL** ?
5. З якими полями застосовують функції **SUM** та **AVG** ?
6. Для чого використовують команду **DISTINCT** у функції **COUNT** ?
7. У чому полягає різниця застосовування команди **ALL** та зірочки (*****) у функції **COUNT** ?

Лекція № 15

ВИВЕДЕННЯ ДАНИХ У МОВІ SQL

Навчальні цілі:

- вивчити використання речень мови SQL для вибору та виведення інформації;
- розглянути порядок виведення запитів.

Навчальні питання:

1. Використання речень для вибору та виведення інформації.
2. Виведення запитів.

15.1 Використання речень для вибору інформації

Агрегатні функції також можна використовувати у реченнях для вибору необхідної інформації.

Речення **GROUP BY** дозволяє визначати підмножину значень в особливому полі у термінах іншого поля, а також застосовувати агрегатну функцію до підмножини. Використання цього речення дає можливість об'єднувати поля і агрегатні функції в єдиному реченні з командою **SELECT**.

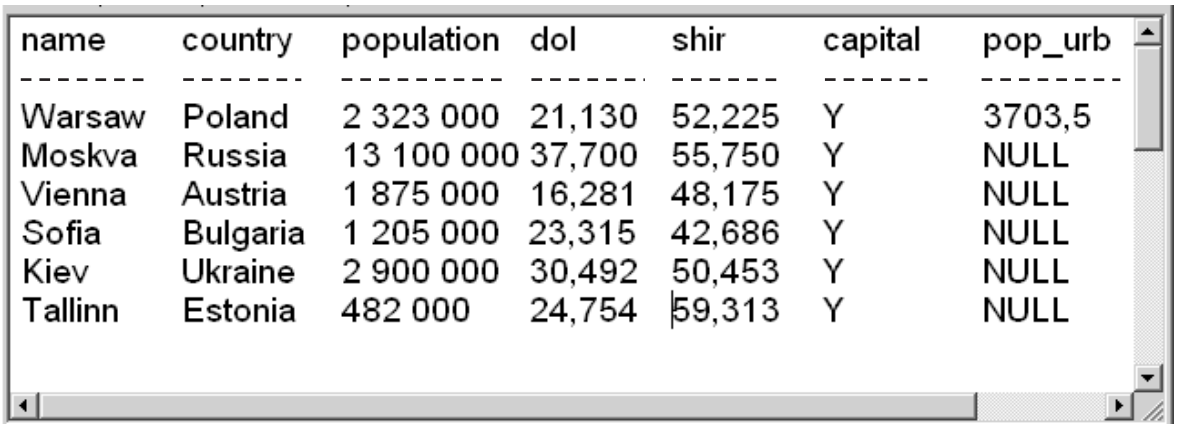
Наприклад, користувачеві необхідно знайти місто з найбільшим населенням у кожній країні з таблиці **town**. Для здійснення цього вибору можна зробити роздільний запит для кожного міста за допомогою агрегатної функції **MAX (population)**. При цьому необхідно перевірити кожне значення поля **country** з таблиці **town**. За допомогою речення **GROUP BY** це ж саме можна здійснити однією командою. Для цього слід набрати на клавіатурі комп’ютера таке:

```
SELECT name, country, MAX (population)
FROM town
GROUP BY country;
```

У результаті виконання команди **SELECT** буде отримано список міст з найбільшим населенням у кожній країні, що містяться у таблиці **town**, як показано на рисунку 15.1.

Необхідно зазначити, що за допомогою речення **GROUP BY** агрегатні функції застосовують для серії груп і визначають значення поля в цілому. В цьому випадку кожна група складається з усіх рядків з тим же самим значенням поля, і агрегатну функцію **MAX** застосовують окремо для кожної такої групи. Поле, до якого застосовують речення **GROUP BY**, має тільки одне значення на групу виведення так само, як і при агрегатній функції. В результаті речення **GROUP BY** дозволяє об'єднувати агрегатні

функції та поля. Речення **GROUP BY** також можна використовувати з декількома полями.



name	country	population	dol	shir	capital	pop_urb
Warsaw	Poland	2 323 000	21,130	52,225	Y	3703,5
Moskva	Russia	13 100 000	37,700	55,750	Y	NULL
Vienna	Austria	1 875 000	16,281	48,175	Y	NULL
Sofia	Bulgaria	1 205 000	23,315	42,686	Y	NULL
Kiev	Ukraine	2 900 000	30,492	50,453	Y	NULL
Tallinn	Estonia	482 000	24,754	59,313	Y	NULL

Рисунок 15.1 – Результат виконання команди **SELECT** із вибору міст з найбільшою кількістю населення у кожній країні з таблиці **town**

Речення **HAVING** визначає критерії, які застосовують, щоб видаляти певні групи з виведення саме так, як команда **WHERE** робить це з окремими рядками.

Наприклад, користувачеві необхідно знайти у таблиці **town** міста з максимальним населенням у кожній країні, але кількість населення має бути більше 1 310 000 людей. У цьому випадку нема можливості використати агрегатну функцію **MAX** з командою **WHERE**, тому що предикати у цій команді оцінюють у термінах одиночного рядка, а агрегатні функції – у термінах груп рядків, тобто буде неправильним, якщо набрати на клавіатурі комп'ютера таке:

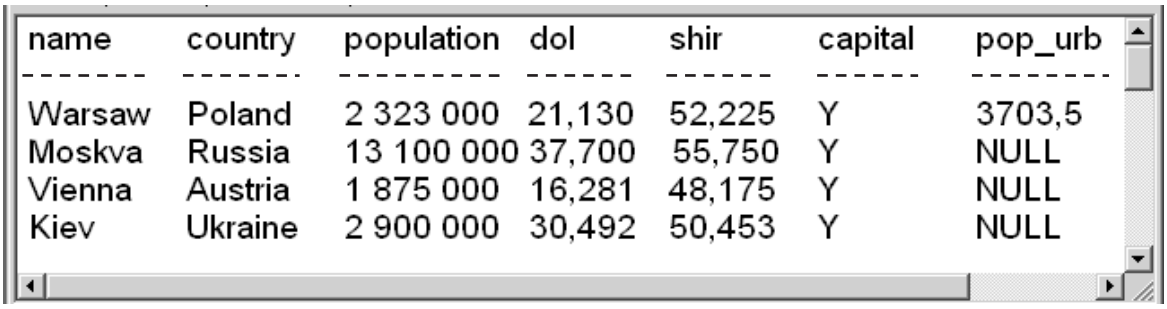
```
SELECT name, country, MAX (population)
FROM town
WHERE MAX (population) > 1 310 000
GROUP BY country;
```

Такий запит буде відхиленням від строгої інтерпретації ANSI, його застосовувати не можна.

Для пошуку міст з максимальним населенням у кожній країні, яке є більшим ніж 1 310 000 людей, можна використовувати речення **HAVING**. Для цього необхідно набрати на клавіатурі комп'ютера таке:

```
SELECT name, country, MAX (population)
FROM town
GROUP BY country
HAVING MAX (population) > 1 310 000;
```

У результаті виконання команди **SELECT** з таблиці **town** буде отримано список міст з максимальною кількістю населення у кожній країні, яке є більшим ніж 1 310 000 людей, як показано на рисунку 15.2.



name	country	population	dol	shir	capital	pop_urb
Warsaw	Poland	2 323 000	21,130	52,225	Y	3703,5
Moskva	Russia	13 100 000	37,700	55,750	Y	NULL
Vienna	Austria	1 875 000	16,281	48,175	Y	NULL
Kiev	Ukraine	2 900 000	30,492	50,453	Y	NULL

Рисунок 15.2 – Результат виконання команди **SELECT** із вибору міст з максимальною кількістю населення у кожній країні, більшою ніж 1 310 000 людей

Необхідно зазначити, що аргументи у реченні **HAVING** застосовують так само, як і у команді **SELECT**, що складається з команд, які використовують у реченні **GROUP BY**. Вони повинні мати одне значення на групу виведення.

15.2 Виведення запитів

Під час роботи з базою даних за допомогою мови SQL користувач може удосконалювати виведення запитів у зручній для нього формі. Наприклад, можна здійснити виведення скалярного виразу за допомогою вибраних полів. Отже, якщо необхідно виконати просте числове оброблення даних, то мова SQL дозволяє поміщати скалярні вирази і константи серед вибраних полів. При цьому скалярні вирази можуть доповнювати або заміщати поля у команді **SELECT**, а також можуть вміщати одне або більше вибраних полів. Надалі дані будуть подані у формі, яка більше відповідає потребам користувача.

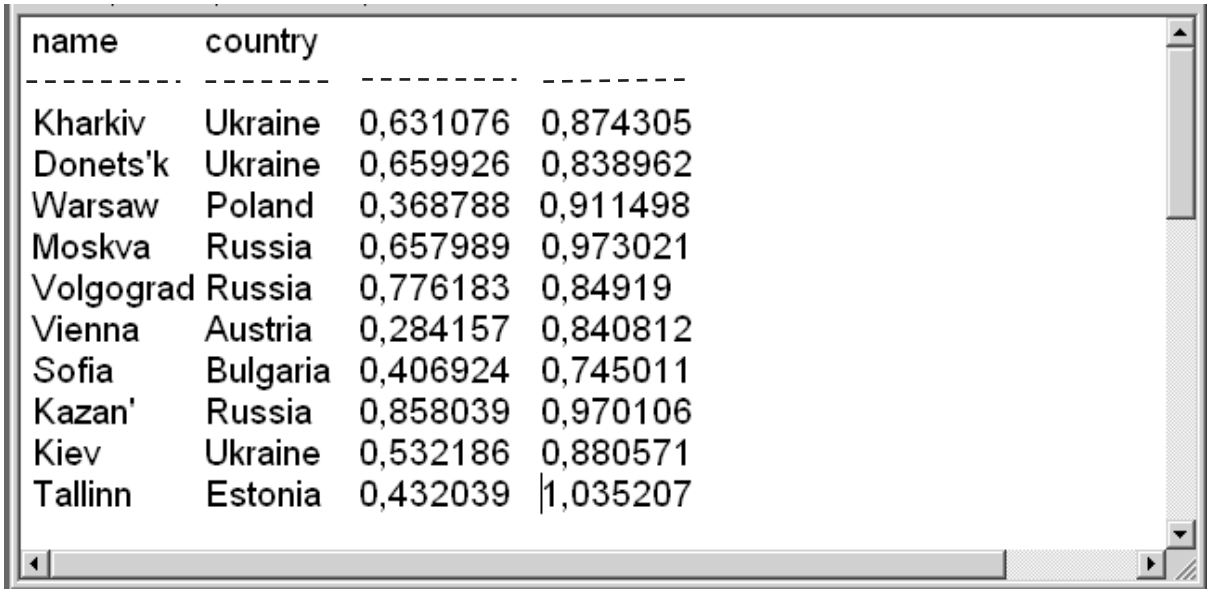
Наприклад, можна навести у таблиці **town** географічні координати міст у радіанах, а не в градусах. Для цього необхідно набрати на клавіатурі комп’ютера таке:

```
SELECT name, country, dol*3,14/180, shir*3,14/180
FROM town;
```

У результаті виконання команди **SELECT** будуть виведені географічні координати міст в радіанах, як показано на рисунку 15.3.

Останні два стовпці, які будуть виводитися, створені у запиті іншим способом, ніж просто вибором їх з таблиці. Ці стовпці називаються стовпцями виведення, які не позначаються, тобто не мають найменування. Стовпці виведення створюються всякий раз, коли використовуються

агрегатні функції, константи або вирази у реченні **SELECT**, також стовпці виведення не мають ніяких імен, оскільки ім'я стовпця є одним з атрибутів таблиці БД.



name	country		
Kharkiv	Ukraine	0,631076	0,874305
Donets'k	Ukraine	0,659926	0,838962
Warsaw	Poland	0,368788	0,911498
Moskva	Russia	0,657989	0,973021
Volgograd	Russia	0,776183	0,84919
Vienna	Austria	0,284157	0,840812
Sofia	Bulgaria	0,406924	0,745011
Kazan'	Russia	0,858039	0,970106
Kiev	Ukraine	0,532186	0,880571
Tallinn	Estonia	0,432039	1,035207

Рисунок 15.3 – Результат виконання команди **SELECT** із виведення географічних координат міст у радіанах

Крім скалярних виразів у виведенні можна розміщувати необхідний текст. Символи, коли вони нічого не означають самі по собі, є константами і їх можна вводити у речення **SELECT**. Однак необхідно зазначити, що символьні константи на відміну від числових констант не можна використовувати у виразах. Наприклад, можна виконати **1 + 2** у реченні **SELECT**, але вираз типу **'A' + 'B'** виконати неможливо, тому що **'A'** та **'B'** це просто букви, а не змінні і не символи. Однак можна вводити просто текст, що зручно, якщо необхідно вивести коментарі або розмірність величин.

Наприклад, для виведення розмірності величин **dol*3,14/180, shir*3,14/180** як тексту – **‘ рад’** необхідно набрати на клавіатурі комп’ютера таке:

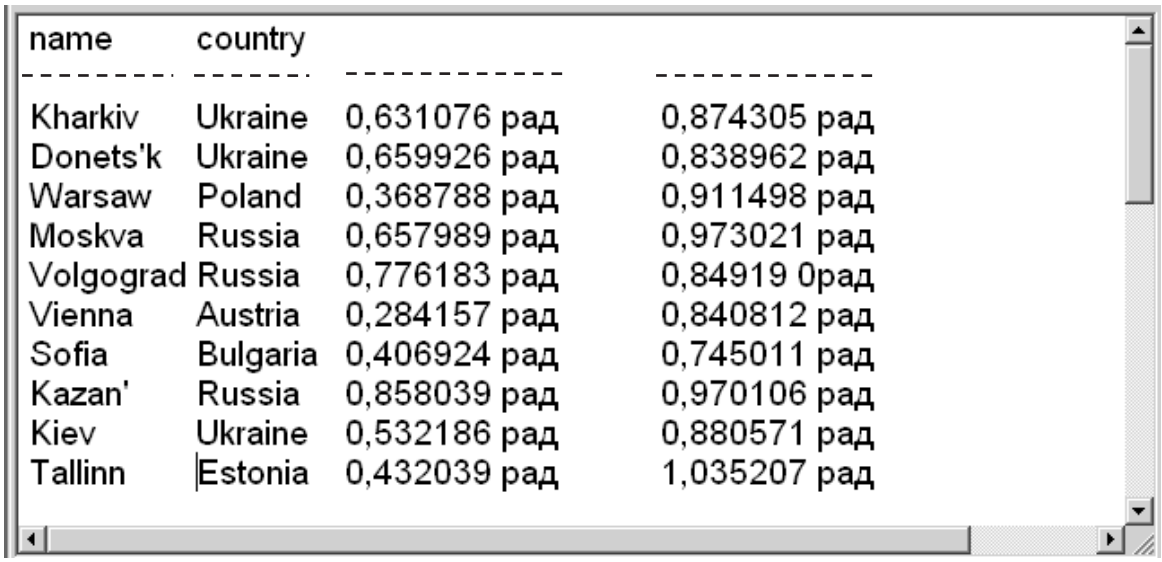
```
SELECT name, country, dol*3,14/180 ‘ рад’, shir*3,14/180 ‘ рад’
FROM town;
```

У результаті виконання команди **SELECT** будуть виведені географічні координати міст у радіанах з коментарем **‘ рад’**, як показано на рисунку 15.4.

Якщо необхідно зробити пропуск після зазначення числа, то слід ввести пропуск перед **‘ рад’** в одиничних лапках, і пропуск буде частиною рядка, що виводиться. Цю особливість можна використовувати для позначення виведення разом з коментарями. Також необхідно зазначити,

що коментар, який було вказано у реченні з командою **SELECT**, буде надрукованим у кожному рядку виводу, а не один раз для всієї таблиці.

Таблиці, що містяться в реляційній БД являють собою невпорядковані набори даних, і дані, які виводяться з них, не обов'язково з'являються у певній послідовності. За допомогою мови SQL користувач може впорядковувати виведення полів. Для цього використовується команда **ORDER BY**.



name	country		
Kharkiv	Ukraine	0,631076 рад	0,874305 рад
Donets'k	Ukraine	0,659926 рад	0,838962 рад
Warsaw	Poland	0,368788 рад	0,911498 рад
Moskva	Russia	0,657989 рад	0,973021 рад
Volgograd	Russia	0,776183 рад	0,84919 0рад
Vienna	Austria	0,284157 рад	0,840812 рад
Sofia	Bulgaria	0,406924 рад	0,745011 рад
Kazan'	Russia	0,858039 рад	0,970106 рад
Kiev	Ukraine	0,532186 рад	0,880571 рад
Tallinn	Estonia	0,432039 рад	1,035207 рад

Рисунок 15.4 – Результат виконання команди **SELECT** із виведенням географічних координат міст у радіанах з коментарем **‘ рад’**

Команда **ORDER BY** дозволяє впорядкувати виведення запиту згідно із значеннями вибраних стовпців. Можна задати виведення даних у порядку зростання за допомогою значення **ASC** або у порядку убування за допомогою значення **DESC** для кожного стовпця. За умовчанням встановлено порядок зростання даних.

Наприклад, необхідно вивести дані з таблиці **town** у порядку убування населення у містах. Для цього необхідно набрати на клавіатурі комп’ютера таке:

```
SELECT *
FROM town
ORDER BY population DESC;
```

У результаті виконання команди **SELECT** буде отримано повний список стовпців таблиці **town** у порядку убування населення у містах, як показано на рисунку 15.5.

Необхідно зазначити, що команду **ORDER BY** можна використовувати одночасно для будь-якої кількості стовпців. Для цього стовпці, які упорядковуються, мають бути вказані у реченні **SELECT**.

Для впорядковування груп команду **ORDER BY** можна використовувати разом з командою **GROUP BY**. При цьому команду **ORDER BY** завжди застосовують останньою.

Під час роботи з базою даних користувач може за допомогою однієї команди мови SQL виводити інформацію з таблиць, які пов’язані певними зв'язками. Виведення такої операції називається об'єднанням і є одним з видів операцій у реляційній БД. Використання операції об'єднання дозволяє пов'язати інформацію з будь-яким числом таблиць. Так створюються зв'язки між фрагментами даних, що порівнюються.

name	country	population	dol	shir	capital	pop_urb
Moskva	Russia	13 100 000	37,700	55,750	Y	NULL
Kiev	Ukraine	2 900 000	30,492	50,453	Y	NULL
Warsaw	Poland	2 323 000	21,130	52,225	Y	3703,5
Donets'k	Ukraine	2 200 000	37,811	48,069	N	4568
Kharkiv	Ukraine	1 940 000	36,158	50,094	N	2384,6
Vienna	Austria	1 875 000	16,281	48,175	Y	NULL
Volgograd	Russia	1 360 000	44,472	48,655	N	NULL
Sofia	Bulgaria	1 205 000	23,315	42,686	Y	NULL
Kazan'	Russia	1 140 000	49,162	55,583	N	NULL
Tallinn	Estonia	482 000	24,754	59,313	Y	NULL

Рисунок 15.5 – Результат виконання команди **SELECT** із вибору повного списку стовпців таблиці **town** у порядку убування населення у містах

Під час об'єднання таблиці, які подано списком у реченні з командою **FROM**, відокремлюються комами. Предикат запиту може посилатися до будь-якого стовпця будь-якої зв'язаної таблиці і може використовуватися для зв'язку між ними. Як правило, предикат порівнює значення у стовпцях різних таблиць, щоб визначити, чи задовольняє значення встановленій умові за допомогою команди **WHERE**.

Для створення об'єднання необхідно набрати на клавіатурі комп’ютера таке:

```
SELECT town.*, country.*
FROM town, country
WHERE town.name = country.cntry_name;
```

При цьому повне ім'я стовпця таблиці складається з імені таблиці, що супроводжується крапкою, а потім імені стовпця. Наприклад, ім'я стовпця таблиці може мати такий вигляд:

town.name
town.country
country.population

Під час створення об'єднання за допомогою мови SQL досліджують кожну комбінацію рядків двох можливих таблиць і перевіряють ці комбінації за їх предикатами. Можна також створювати запити, об'єднуючи більше двох таблиць.

Запитання для самоперевірки

1. Для чого використовують речення **GROUP BY** ?
2. У чому полягає різниця між командами **HAVING** та **WHERE** ?
3. Для чого використовується речення **HAVING** ?
4. Як здійснюється виведення скалярного виразу за допомогою вибраних полів ?
5. Як здійснити коректне виведення розмірності числових величин ?
6. Яку команду використовують для впорядкування виведення запиту у порядку зростання ?
7. Як вирізняються повні імена стовпців таблиць, що пов’язані між собою ?

Лекція № 16

ПОБУДОВА МОДЕЛІ ДАНИХ БАЗИ ГЕОДАНИХ

Навчальні цілі:
- вивчити принципи побудови моделей геоданих;
- розглянути етапи створення моделі даних.

Навчальні питання:
1. Принципи побудови моделей геоданих.
2. Етапи створення моделі даних.

16.1 Принципи побудови моделей геоданих

Поняття «моделювання даних» – це переклад з англійської data modeling. Необхідно зазначити, що сам переклад не розкриває суті цього поняття, тому що безпосередньо моделюються не дані, а моделюється реальність за допомогою даних. Отже, модель даних – це модель реальності, яка відображається у структурі даних і утворюється взаємозв'язками елементів даних.

Таким чином, прикладні моделі даних є віртуальною імітацією об'єктів управління або дослідження. За допомогою таких моделей можна відтворювати властивості та відносини реальних об'єктів, а також їх поведінку. Крім того, за допомогою цих моделей можна отримувати інформацію про модельовані об'єкти, аналізувати їх взаємодію та досліджувати різні сценарії їх розвитку. Найбільший виграш при цьому досягається, якщо необхідно обробляти дані про тисячі та навіть мільйони реальних об'єктів.

У сучасних ГІС для моделювання поверхонь можна використовувати три типи моделей: растрову, векторну та модель на базі нерегулярної тріангуляційної мережі (TIN). Растрова та векторна моделі просторових даних дозволяють описати те, що знаходиться на поверхні Землі двома різними способами.

Растрова модель дозволяє вказати тільки окремі характеристики в окремих точках простору. При цьому ці точки розподілені регулярно по всій поверхні. Детальність такого уявлення визначається кроком сітки (або розміром чарунки). У чарунках можуть зберігатися як якісні ознаки (наприклад, класи – водна поверхня, ліс, рілля, місто і т.д.), так і кількісні (температура, вологість, оптична яскравість і т.д.). У першому випадку це – так званий тематичний растр, у другому – півтоновий.

За допомогою векторної моделі даних виділяють об'єкти на поверхні землі і наводять опис кожного з них. Такий підхід також використовують у традиційній картографії, де досліджують так звані об'єкти картографування і правила їх відображення на картах.

Векторна модель має розширення (тобто надбудови над базовою моделлю), такі, як топологія та тріангуляційні мережі. Топологію поділяють на лінійно-вузлову (для зображення мереж) і полігональну (для подання суцільних покриттів). Розвиток ГІС привів до їх інтеграції з технологією реляційних баз даних, результатом якої стало не тільки впровадження формальних методів моделювання даних, але також і клієнт-серверної технології з багатокористувальним редагуванням і підтримкою множинних версій даних.

Модель даних бази геоданих фактично забезпечує зв'язок між сприйняттям людиною реальних об'єктів, що її оточують, та поданням (збереженням) цих об'єктів у реляційних базах даних.

Традиційно проектування реляційної бази даних містить два основні етапи – **з'ясування логічної моделі даних** та **фізична реалізація моделі** (або схеми) бази даних. За допомогою логічної моделі даних фіксується уявлення даних користувачем, а фізична реалізація моделі бази даних забезпечує подання цих даних технологічними засобами реляційних баз даних.

Під час проектування логічної моделі даних необхідно точно визначити набір об'єктів, які є необхідними користувачеві, а також як вони відносяться між собою. Наприклад, об'єктами, які необхідно моделювати, є вулиці, земельні ділянки, власники або будинки, а прикладами відношень між ними можуть бути: «будівля розташована на земельній ділянці», «земельна ділянка знаходиться у власності Петрова» або «вулиця є частиною району».

Створена початкова логічна модель даних дозволяє перевірити відповідність її вимогам користувача щодо введення даних, їх оновлення або доступу до них, а також дає можливість випробувати створену модель під час виконання різних робочих процедур та правил, які встановлені в організації.

Під час логічного проектування необхідно враховувати вимоги всіх груп користувачів організації, а не базуватися на поглядах тільки однієї групи, тому що модель даних у цьому випадку буде обов'язково містити недоліки з погляду інших неврахованих груп користувачів. Якість побудованої логічної моделі даних визначається практичним досвідом фахівців, що її створюють.

Після побудови логічної моделі необхідно оцінити, наскільки створена модель даних є наближеною до бажаного результату. При цьому оцінюють відповідність логічної моделі даних правилам, що встановлені в організації, різне подання даних, які є необхідними різним групам користувачів, а також наявність дублювання даних у моделі.

Існує декілька способів подання логічних моделей даних, найчастіше у вигляді так званих діаграм суть-зв'язок (entity-relationship diagrams, ER-diagrams). Для реалізації цього підходу були запропоновані різні методики проектування та системи позначень на цих схемах, що враховують потоки

даних або сценарії використання даних. Основним недоліком ER-діаграм є те, що їх зовнішній вигляд змінюється залежно від методики проектування.

У наш час більшість розробників об'єктно-орієнтованих моделей для подання об'єктних моделей використовують уніфіковану мову моделювання (Unified Modeling Language, UML). Необхідно зазначити, що мова UML не є методикою проектування, а є системою зображення за допомогою діаграм. Можна подавати модель стандартним способом з використанням будь-якої методики об'єктно-орієнтованого проектування, користуючись мовою UML.

Реалізація фізичної моделі бази даних здійснюється на основі логічної моделі даних, яку отримує фахівець з реляційних БД від розробника моделі даних. Після цього фахівець з реляційних БД з використанням інструментів адміністрування СУБД задає схему БД і створює нову БД, яка є готовою до прийняття даних. Фізична модель бази даних будується за схожими правилами, як це робиться під час побудови логічної моделі даних. При цьому класи об'єктів можна поділяти або об'єднувати при їх реалізації у таблицях, а також правила і відношення можуть бути подані різними способами.

Після побудови логічної моделі даних та її фізичної реалізації створюють базу геоданих. Процес проектування реляційної бази геоданих показано на рисунку 16.1. Основною перевагою бази геоданих є те, що хоча вона являє собою фізичну реалізацію даних, але й дозволяє користувачеві структурувати дані так, як це робиться у логічній моделі даних.

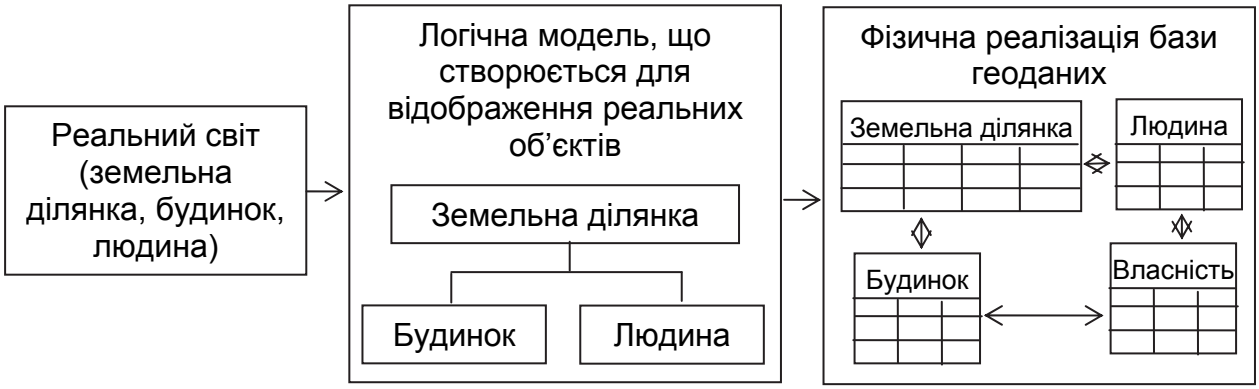


Рисунок 16.1 – Процес проектування реляційної бази геоданих

Як видно з рисунка 16.1, логічна модель даних є відображенням реальних об'єктів, які зустрічаються у реальному світі, а це подання далі перетворюється в елементи бази геоданих. При цьому об'єкт являє собою якусь суть реального світу, наприклад, будинок, земельна ділянка або людина (клієнт). У базі геоданих об'єкт зберігається у вигляді рядка таблиці та має набір атрибутів. Зазначені атрибути характеризують якості об'єкта, такі, як назва, розмір, категорія, або зберігають ідентифікатори

(ключі) інших об'єктів. Атрибути зберігаються у стовпцях (полях) таблиць бази геоданих. У свою чергу, набір подібних об'єктів створює клас даного об'єкта. Всі об'єкти у класі мають однаковий набір атрибутів. Клас об’єкта зберігається у базі геоданих у вигляді таблиці. Рядки та стовпці таблиці утворюють двовимірну матрицю.

Необхідно зазначити, що географічні бази даних містять складні дані, такі, як контури лінійних і площадкових просторових об'єктів. Ці об’єкти являють собою структуровані набори координат, які важко описати за допомогою полів атомарного типу, наприклад, цілим або дійсним числом. Крім того, просторові об'єкти об'єднують у системи, що містять явні топологічні, неявні просторові або загальні відношення.

Таким чином, основна спрямованість бази геоданих полягає в обробленні складних географічних даних за допомогою однорідної моделі даних, яка є не залежною від вибраної реляційної бази даних. Основою для створення бази геоданих є реляційна база даних.

16.2 Етапи створення моделі даних

Моделі даних можна розробляти як для вирішення задач індивідуального користувача, так і великих організацій (корпоративних або багатокористувальних систем). У другому випадку створення моделі даних є складним завданням. Це обумовлено тим, що корпоративна інформаційна система має відображати все різноманіття аспектів діяльності організації. Отже, фактично, модель даних корпоративної інформаційної системи має бути моделлю самої організації.

Необхідно зазначити, що після створення та початку експлуатації інформаційної системи в організації, внесення змін у її модель даних стає складним і дорогим завданням. Це пов’язано з тим, що необхідно припинити роботу з системою, проаналізувати і протестувати великий обсяг даних. При цьому доводиться узгоджувати всі зміни моделі даних. Отже, створення моделі даних складних систем є важливим завданням.

Процес створення моделі даних можна поділити на декілька етапів:

- визначення кола користувачів системи;
- створення концептуальної моделі даних;
- побудова логічної моделі даних;
- розроблення фізичної моделі даних;
- створення екземпляра бази даних та його тестування;
- заповнення бази даних актуальними даними.

На кожному етапі розробники вирішують певні завдання (часткові) з використанням початкових даних та отримують проміжні результати на виході кожного етапу. Коротко розглянемо зміст основних етапів створення моделі даних.

На першому етапі визначається коло користувачів системи, завдання, які вони мають вирішувати за її допомогою. Крім того,

визначаються інформаційні продукти, які має створювати система, а також дані, які для цього необхідні. Отже, на цьому етапі формуються вимоги до інформаційної системи.

На другому етапі створюється концептуальна модель даних, яка містить основну їх сутність та основні зв'язки між ними (без визначення типу). На цьому етапі визначаються всі об'єкти, які необхідно моделювати, та зв’язки між ними. При цьому під час створення концептуальної моделі не потрібно ставити кожну сутність у відповідність якомусь класу об'єктів у фізичній базі даних. Наприклад, будь-яка адреса є цілком самостійним об'єктом, що існує незалежно від того, є там будівля чи ні. А в каталозі об'єктів будівництва адреса стає вже атрибутом таких об'єктів. На цьому рівні абстрагування атрибути (характеристики) об'єктів не є істотними, і в концептуальній моделі даних вони не відображаються, хоча деяка сутність може надалі виявитися атрибутами, так це буде у прикладі з адресою.

На третьому етапі будується логічна модель даних. На цьому етапі показуються всі атрибути об'єктів та визначаються типи їх відношень. За допомогою логічної моделі даних не конкретизуються механізми реалізації, які будуть використовуватися у СУБД. При цьому не є важливим, підтримує СУБД складені ключі чи ні, є в ній спеціальний тип даних «дата-час» чи ні. Якщо у вимогах до створюваної інформаційної системи не вказано, яку конкретно СУБД слід використовувати, то в результаті логічного проектування можна сформулювати набір вимог до СУБД, на основі яких можна вже вибирати конкретну систему.

На четвертому етапі розробляється фізична модель даних, в якій існують всі класи (таблиці), які мають бути створені у базі даних, а також указуються механізми реалізації відношень та елементи поведінки об'єктів. Наприклад, відношення між класами об'єктів можна реалізовувати як за допомогою зовнішнього ключа, тобто зберігання ідентифікатора-посилання в окремому полі таблиці одного з класів, так і за допомогою класу відношень – окремої таблиці, що містить ідентифікатори об'єктів двох класів, які поєднані. Після створення фізичної моделі даних стає зрозуміло, яка частина функціональності інформаційної системи має відношення до базового програмного забезпечення, а яку слід написати програмістам проекту.

На п'ятому етапі створюється екземпляр бази даних і проводиться його тестування, що дозволяє оцінити правильність вибору того або іншого рішення стосовно фізичної моделі даних. Якщо вибраний варіант рішення не приводить до належної продуктивності, фізичну модель даних може бути скореговано. Паралельно створюється програмний код, що реалізує функції інформаційної системи, та проводиться його тестування на пробному екземплярі бази даних.

На шостому етапі база даних заповнюється актуальними даними, і система запускається в роботу.

Необхідно зазначити, що на кожному з етапів можуть виявлятися особливості, які потребують змін окремих рішень, прийнятих на попередньому етапі. Крім того, складні системи можна розбивати на компоненти, для реалізації кожного з яких також можна застосовувати цю ж послідовність етапів. При цьому створення моделі даних зводиться до повторення зазначених вище етапів, а сам процес стає ітеративним.

Таким чином, створені моделі бази даних дозволяють будувати ефективні бази геоданих. При цьому є можливість використовувати готові функції перевірки та підтримки правил топології, контролю введення даних засобами атрибутивних доменів, широкий спектр розширень базових моделей даних (мережних, геодезичних, схематичних і т.д.), класи відношень, множинні картографічні подання та багато іншого.

Запитання для самоперевірки

1. У чому полягає сенс поняття "моделювання даних" ?
2. Які існують типи моделей даних сучасних ГІС та в чому полягає їх відмінність ?
3. Які основні етапи містить проектування реляційної бази даних ?
4. У чому полягає різниця між логічною та фізичною моделями бази даних ?
5. Як будується логічна модель даних ?
6. Як створюється фізична модель даних ?
7. Які існують етапи створення моделі даних та в чому полягає сенс цих етапів ?

Лекція № 17

ПРИНЦИПИ ПРОЕКТУВАННЯ БАЗИ ГЕОДАНИХ

Навчальні цілі:
- вивчити принципи проектування бази геоданих;
- розглянути етапи проектування бази геоданих.

Навчальні питання:
1. Основні принципи проектування бази геоданих.
2. Основні етапи проектування бази геоданих.

17.1 Основні принципи проектування бази геоданих

У зв’язку з тим, що база геоданих (БГД) є різновидом реляційної бази даних, яка містить додаткову структуру для подання географічних даних, то проектування БГД проводиться так само, як і проектування будь-якої бази даних. При цьому застосування БГД приводить до розширення та спрощення процесу проектування БГД завдяки об'єктно-орієнтованій структурі даних, які відображають просторові й топологічні відношення географічних об'єктів. Об'єктно-орієнтована структура БГД дозволяє створювати спеціальний механізм для подання наборів об'єктів у вигляді інтегрованих систем, таких, як річкові або дорожні мережі, а також набори земельних ділянок. Ця структура реалізується всередині набору класів об'єктів і називається топологією.

Нагадаємо, що структурно база геоданих складається з набору географічних даних, класів просторових об'єктів, об'єктних класів та класів відношень. Всі зазначені елементи дозволяють проектувати географічні бази даних, які є близькими до своїх логічних моделей даних.

У проекті бази геоданих реалізується географічна модель даних, яка є основою для всіх операцій з ГІС, таких, як створення змістовних карт, пошук інформації та виконання просторового аналізу.

Процес проектування БГД є подібним до процесу створення будь-якого іншого проекту. Він починається з розуміння мети, розвивається з розширенням рівнів детальності в результаті збору інформації та наближається до кінцевої реалізації. Наприклад, проектування транспортної моделі починається з вивчення існуючого потоку дорожнього руху. Далі проектування здійснюється у вигляді ряду кроків відповідно до зростаючої детальності моделі, наприклад, врахування особливостей приросту населення. Проект створюється, починаючи від попереднього плану та кошторису і закінчуючи детальними технічними кресленнями.

На початку проектування бази геоданих необхідно ретельно продумати призначення та основну мету проекту БГД. При цьому враховуються можливості географічної інформаційної системи, що є в

організації, та завдання, які виконуються щоденно та під час тривалого планування. Крім того, враховуються способи реалізації різних функцій, зберігання та сумісного використання даних підрозділами організації, а також можливості інтеграції з іншими технологіями.

На початковій стадії створення проекту бази даних також визначається необхідність та доцільність проектування. Під час виконання цього етапу з'ясовуються дані, які є найбільш необхідними та важливими для організації, порядок та можливості збереження даних у базі геоданих, можливі технології для реалізації ГІС, які дозволять раціонально використовувати існуючі функції або змінювати спосіб досягнення конкретної мети. Крім того, визначається відповідальний за підтримку бази геоданих.

Таким чином, процес створення проекту БГД містить:

- визначення мети проекту;
- визначення, аналіз та оцінювання варіантів проекту;
- узгодження плану його реалізації.

Основними принципами проектування БГД є:

1. Під час проектування необхідно залучати користувачів БГД. За допомогою користувачів є можливість отримати необхідні відомості для проектування бази геоданих. При цьому вони зроблять свій внесок у загальну справу створення якісної БГД.

2. Розроблення БГД має просуватися послідовно крок за кроком. Необхідно зазначити, що процес проектування є інтерактивним та ітеративним, тому немає сенсу відразу створювати повний детальний проект. Доцільно виконувати проектування за стадіями відповідно до потреб організації.

3. Для ефективного проектування необхідно створити команду розробників. Члени команди повинні мати різноманітні навички та знання, які можуть знадобитися під час виконання проекту, а також вміти приймати рішення. Різні фахівці можуть залучатися до команди на різних стадіях створення проекту.

4. Для якісного виконання проекту БГД необхідно застосовувати творчий підхід, для чого доцільно проводити аналіз новітніх технологій та вибирати найбільш придатні для реалізації в існуючих ГІС, що застосовуються для вирішення завдань конкретної організації.

5. Під час проектування доцільно створювати окремі елементи (блоки) проекту. При цьому весь великий проект поділяється на окремі та чітко визначені робочі блоки. Для кожного блока необхідно встановлювати часові межі та проміжні звіти (не частіше одного разу в два місяці). Така організація дозволить здійснити ефективне управління та чітку спрямованість проекту.

6. Під час виконання проекту необхідно постійно пам'ятати про мету та завдання організації, для якої його розробляють. Проектування та його

реалізація спрямовані на виконання реальних вимог, які були поставлені організацією та користувачами.

7. На початку роботи над проектом не потрібно деталізувати завдання. Деталі проекту додаються на відповідних етапах його розроблення. Наприклад, не слід визначати всі правила перевірки достовірності класів просторових об'єктів раніше безпосереднього створення бази геоданих. Протягом виконання проекту нові деталі його розроблення додаються на початку кожного етапу роботи, щоб команда могла поступово переходити до наступної стадії.

Таким чином, проектування бази геоданих забезпечує її повноцінну архітектуру. Проектування дозволяє розробникові уявляти базу даних в цілому та оцінювати взаємозв'язок її різних аспектів. Витрати часових та матеріальних ресурсів на формулювання та вирішення питань проектування забезпечують якісне його виконання та виключають появу значних помилок у подальшому.

Результати проектування БГД є позитивними, якщо розроблена база даних задовольняє таким вимогам:

- відповідає поставленій меті та організаційним питанням;
- містить всі необхідні дані та не містить надлишкових даних;
- подає дані так, щоб користувачі могли працювати з ними у багатокористувальному режимі;
- допускає різне подання даних;
- дозволяє працювати з додатками, що підтримують дані, та з додатками, що їх використовують;
- надає можливість якісного подавання, кодування та створення географічних об'єктів.

У результаті якісного проектування БГД буде досягнуто:

- збільшення гнучкості у пошуку даних та їх аналізі;
- зростання можливості розроблення додатків користувачами;
- якісне розподілення вартості збирання даних, їх зберігання та використання;
- подання даних у зручному вигляді, що забезпечує різноманіття способів їх використання;
- постійне оновлення даних, до яких звертається багато різних користувачів;
- розширення функціональних можливостей даних;
- мінімальне введення надлишкових даних.

17.2 Основні етапи проектування бази геоданих

Процес проектування бази геоданих можна подати як послідовність виконання декількох етапів.

Основними етапами проектування бази геоданих є:

1. Моделювання уявлень користувача.

2. Визначення об'єктів та їх відношень.

3. Вибір подання об'єктів.

4. Узгодження з моделлю даних бази геоданих.

5. Організація БГД у вигляді наборів географічних даних.

На перших трьох етапах розробляють концептуальну модель БГД. Просторові об'єкти класифікують залежно від їх ролі у забезпеченні функцій організації. При цьому вирішують питання про їх просторове подання (точка, лінія, площадковий об'єкт, зображення, поверхня або негеографічний об'єкт).

На четвертому та п’ятому етапах розробляють логічну модель даних, яка пов'язує концептуальні моделі з наборами географічних даних.

На першому етапі моделюють уявлення даних користувачами. Спочатку розробник проводить опитування користувачів, з'ясовує структуру і функції організації та аналізує вимоги, що ставляться. З'ясування функцій організації є необхідним для того, щоб забезпечити підтримку цих функцій. На цьому етапі також упорядковують дані у логічну структуру.

Другий етап характеризується визначенням (виділенням) об'єктів, їх описом та встановленням відношень між ними. При цьому формують логічну модель даних з набором об'єктів та їх взаємозв'язками (відношеннями), яка зберігається у вигляді схеми.

На третьому етапі вибирають географічне подання об’єктів. При цьому визначають, яке із зображень найбільш відповідає вибраним даним – у вигляді векторів, растру, TIN або місцеположень. Дискретні об'єкти доцільно відображати у вигляді точок, ліній та полігонів, безперервні явища – за допомогою растрів, а поверхні моделюють за допомогою TIN або растрів.

На четвертому етапі узгоджують базу геоданих з моделлю даних. При цьому визначають геометричні типи дискретних просторових об'єктів та відношення між ними, а також типи атрибутів об'єктів. Крім того, на цьому етапі встановлюють відповідність між елементами бази геоданих та відображають об'єкти логічної моделі даних в елементах бази геоданих.

П’ятий етап проектування бази геоданих призначено для формування структури бази геоданих з урахуванням тематичних груп, топологічних асоціацій та відповідальності підрозділів організації за подані дані. При цьому створюють систему просторових об'єктів та встановлюють топологічні зв'язки, задають систему координат і визначають відношення та правила.

Необхідно зазначити, що під час проектування БГД на кожному з етапів необхідно проводити ретельне документування. Необхідність документування зростає зі збільшенням складності конфігурації проекту. Створення робочих схем дозволяє уявити процес об'єднання всіх елементів проекту.

Слід мати на увазі, що під час проектування заключний варіант проекту БГД буде відрізнятися від початкового. Це пов'язано з тим, що проект постійно удосконалюють у зв'язку з більш детальним вивченням та зміною завдань організації, впровадженням нових технологій та опануванням їх розробниками. Тому процес проектування БГД має бути гнучким.

Крім того, на етапі остаточного виконання проекту БГД необхідно розробити план його реалізації, який має спиратися на логічну модель. Такий план спрямовується на забезпечення ключових пріоритетів організації, для якої розробляють БГД. При цьому спочатку створюють додатки для управління даними, а потім – нові набори даних.

Запитання для самоперевірки

1. У чому полягає процес проектування бази геоданих ?
2. Які принципи покладені в основу проектування бази геоданих ?
3. Коли результати проектування бази геоданих вважають задовільними ?
4. Що забезпечує якісне проектування бази геоданих ?
5. Які існують основні етапи проектування бази геоданих ?
6. На яких етапах проектування бази геоданих розробляють концептуальну модель БГД ?
7. Які існують особливості проектування бази геоданих ?

Лекція № 18

КРОКИ ПОБУДОВИ БАЗИ ГЕОДАНИХ

Навчальні цілі:

- вивчити основні кроки побудови бази геоданих;
- розглянути особливості роботи на кожному з етапів проектування бази геоданих.

Навчальні питання:

1. Моделювання уявлень користувача.
2. Визначення об'єктів і відношень.
3. Вибір подання об'єктів.
4. Узгодження з моделлю даних бази геоданих
5. Створення БГД у вигляді наборів географічних даних

Процес проектування бази геоданих можна подати як послідовність виконання декількох етапів. Розглянемо більш докладно особливості роботи розробників на кожному з етапів проектування БГД.

18.1 Моделювання уявлень користувача

Основною метою на цьому етапі є забезпечення загального уявлення та розуміння проекту як фахівцями, що його створюють, так і користувачами, які будуть безпосередньо використовувати базу геоданих під час роботи.

На цьому етапі розробники БГД виконують такі кроки:

- з’ясовують функції, які відповідають цілям та завданням організації, яка буде використовувати БГД;
- визначають дані, які є необхідними для забезпечення визначених вище функцій;
- упорядковують дані у логічні набори об'єктів;
- розробляють початковий план реалізації проекту;
- визначають організаційні функції.

Коротко розглянемо суть цих кроків.

З'ясування функцій, які виконуються в організації, є спрямованим перш за все на визначення цілей та завдань організації. Визначені функції є відправною точкою проектування бази геоданих. На цьому кроці розробники БГД працюють саме з цими функціями, а не з підрозділами організації, тому що функції змінюються не так часто, як структура організації. Всередині організації певні функції можуть виконуватися різними підрозділами у різні часові інтервали, тобто функції організації залишаються незмінними, а виконавці цих функцій можуть змінюватися.

На початку проектування БГД розробники визначають кожну з функцій організації, ретельно її описують, тобто визначають дії, які виконує кожна функція. Наприклад, такими діями можуть бути: управління процесом розвитку земель; контроль використання земель; розроблення угод про будівництво комунікацій.

Під час визначення функцій організації інформаційними джерелами можуть бути як робітники організації (користувачі), так і документація та карти. При цьому розробники мають використовувати загальні публікації, плани стратегічного розвитку організації та плани інформаційних систем.

На рисунку 18.1 показано з'ясування функцій організації, що займається управлінням розвитку земель.

Рисунок 18.1 – Приклад з'ясування функцій організації, що займається управлінням розвитку земель

На другому кроці визначають дані, які необхідні для виконання наведених вище функцій. При цьому слід з'ясувати, яка з функцій створює дані, а яка – використовує вже створені дані. Як правило, дані, які

надходять від певної функції, були створені цією функцією. Це вказує на те, що така функція відповідає за визначення, збір, зберігання та розподіл цих даних. Дані, які надходять до функції, як правило, спрямовані від іншої функції. Дані, що отримані від зовнішньої організації, можна зберігати і управляти ними усередині окремої функції. Обміни даними можуть відбуватися у багатьох формах, наприклад, запити даних і відповіді на запити та ін.

Необхідно також визначити, які дані безпосередньо належать до сфери інтересів організації, а які є фоновими, що забезпечують роботу з основними даними. Розробникам під час проектування БГД слід більш ретельно моделювати основні дані організації.

Для розглянутого вище прикладу щодо організації, що займається управлінням розвитку земель, дані, які є необхідними для забезпечення певних функцій організації, можна подати у вигляді таблиці, наприклад, як таблиця 18.1.

Таблиця 18.1 – Дані, необхідні для забезпечення функцій організації

Записи про землю	
Тип даних	Джерело даних
Земельна ділянка	Плани відведення ділянки
Опис ділянки	Земельний кадастр
Фотознімок ділянки	Історичний архів
Власник ділянки	Право власності на земельну ділянку
Адреса	База даних телефонів

На цьому кроці розробники проводять аналіз сфер дій кожної функції, а також досліджують взаємодію функцій і зовнішніх учасників. При цьому встановлюють з ким або чим взаємодіє конкретна функція та який характер має ця взаємодія. Для кожної функції визначають всі типи даних, які є необхідними, щоб виконувати вимоги щодо переведення інформації. У свою чергу для кожного типу даних визначають найімовірніше джерело даних.

За допомогою співвідношення даних з функціями, які їх створюють та зберігають, розробники виявляють синоніми та функції, що дублюють збір та зберігання даних. У разі виявлення такої ситуації необхідно її вирішити негайно або, принаймні, записати у протокол для майбутнього її усунення.

Для якісного виконання даного кроку є необхідним спілкування з тими, хто виконує ті чи інші функції, тому що саме вони знають, що роблять, з ким взаємодіють та якою інформацією обмінюються. Після того, як всі необхідні дані будуть задокументовані, розробники надають користувачам можливість перевірити створену діаграму і текст, що її супроводжує.

Третій крок полягає в упорядковуванні даних у логічні набори об'єктів. При цьому проводять групування всіх даних на верхньому рівні, тобто даних, які, як передбачається, будуть взаємодіяти з БГД, що розробляють. Створені групи можуть являти собою системи різного типу, наприклад, інженерні комунікації водопостачання, земельні ділянки, вулиці, рельєф та ін. Кожну із створених груп слід використовувати з конкретною функцією з метою отримання або передавання інформації. Наприклад, за допомогою моделі поверхні з певною кількістю опадів можна передати гідрологічні дані у модель річкової мережі. Кожній з цих груп слід мати загальну систему координат, топологічний тип (мережний, плоский, або ніякий). В цілому ці групи мають взаємодіяти.

Для прикладу, що розглядали вище, логічними наборами об'єктів можуть бути: записи про землю, рельєф, вулиці, газові комунікації.

Таким чином, за допомогою опису всіх типів географічних даних, які підтримує організація, виділяється обмежений набір груп, які охоплюють усі системи географічних даних.

На наступному кроці розробляють початковий план реалізації проекту. При цьому на базі логічної моделі створюють план реалізації, який спрямовано на забезпечення ключових пріоритетів організації, для якої створюється БГД. Частина проектного плану має містити кошторис витрат на збирання, оброблення та перевірку даних.

На останньому кроці першого етапу проектування БГД визначають організаційні функції, тобто розподіляють обов’язки між членами команди, яка здійснює проектування БГД. Певні фахівці (розробники) можуть відповідати за різні сегменти проектування БГД.

Останні два кроки спрямовані на вдосконалення способів роботи членів команди та збільшення вигоди від реалізації проекту, що розробляють.

18.2 Визначення об'єктів і відношень

На попередньому етапі було проведено класифікацію функцій, даних і відношень між ними. На другому етапі проектування БГД продовжується детальне дослідження та класифікація даних. При цьому виділяються об'єкти, які можна чітко визначити та які мають загальний набір властивостей. Такі об'єкти називаються примітивами.

На цьому етапі розробники БГД виконують такі кроки:

- визначають та описують примітиви;
- визначають та описують відношення між цими примітивами;
- здійснюють документування примітивів і відношень за допомогою діаграм UML.

Коротко розглянемо суть цих кроків.

На перших двох кроках цього етапу розробники мають чітко визначити примітиви та відношення між ними, які ідентифікуються за

допомогою інтерпретації їх формулювань. Під час формулювання іменники, як правило, належать до примітивів, тоді як дієслова визначають відношення між примітивами. Наприклад, формулювання «клапан управляє рухом газу» описує примітив, а формулювання «газовий пристрій з'єднується з одним або більшою кількістю газопроводів» визначає структурні відношення між примітивами. Формулювання «газова система складається з газового пристрою та газопроводів» описує об'єднання примітивів, що створюють новий, складніший примітив, а «магістральний газопровід – це тип газопроводу» визначає підклас примітивів.

Таким чином, під час визначення об'єктів (примітивів) і відношень між ними доцільно утворювати вислови, які описують об'єкти та їх поведінку. Іменники – це об'єкти, а дієслова – це відношення.

Цей крок також можна виконати, якщо записати перелік послідовних формулювань, що починається словами «система газопостачання складається з таких пристроїв і газопровідних ліній». Кожне формулювання має бути простим і точним. Наприклад, вентиль контролює витікання газу; розподільний пристрій, сполучений з однією або більшою кількістю магістралей; газопровідна магістраль – це є тип газопровідної лінії.

Під час визначення відношень між об'єктами необхідно враховувати, що у багатьох об'єктів є відношення, які схожі з іншими об'єктами. Характеристика відношень визначає проектування бази геоданих.

На третьому кроці здійснюється документування примітивів та відношень за допомогою створення простих діаграм UML. Рекомендується документувати проект з використанням програмного забезпечення для ділової графіки типу Visio. На діаграмі UML прямокутники являють собою примітиви, а лінії – відношення.

На рисунку 18.2 зображено частину діаграми UML, що стосується газових комунікацій.

Із цього рисунка видно, що на діаграмі UML відображено таке:

- лінія газопроводу – це тип лінії газової мережі;
- магістраль та лінія відгалуження утворюють тип лінії газопроводу;
- магістраль може бути пов'язана з багатьма захисними лініями або ні з одною;
- захисна лінія може бути пов'язана з єдиною магістраллю або ні;
- напірні магістралі 1 та 2 є типами магістралей.

Таким чином, на другому етапі проектування застосовують детальне подання даних, з якими працюють користувачі. На цьому етапі слід залучити користувачів до визначення та перевірки правильності створених моделей. Крім того, під час роботи з великою кількістю даних доцільно їх поділити на блоки, які відповідають одночасно тільки одній функції. Необхідно зазначити, що для визначення примітивів, відношень та їх уточнення може знадобитися декілька повторів (ітерацій).

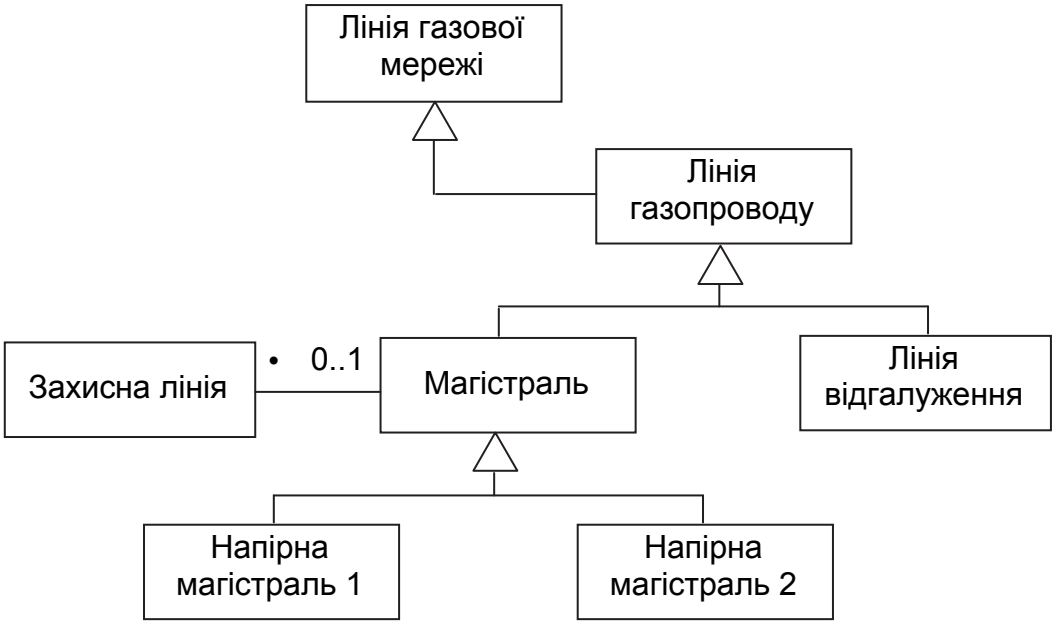


Рисунок 18.2 – Частина діаграми UML газових комунікацій

18.3 Вибір подання об'єктів

На третьому етапі проектування БГД розробники здійснюють класифікацію примітивів за типами їх подання. Деякі примітиви матимуть геометричне зображення з відповідними атрибутами. Такі примітиви класифікують відповідно до їх геометричних характеристик. Інші примітиви будуть подані тільки в алфавітно-цифровому вигляді, а деякі – у вигляді зображень, фотографій або креслень.

На цьому етапі розробники БГД виконують такі кроки:

- вибирають географічне подання об’єктів;
- здійснюють просторове зображення об’єктів (векторне, растрове або за допомогою TIN).

Під час вибору географічного подання об’єктів (перший крок третього етапу) розробники мають враховувати таке:

- можна або не можна зобразити просторовий об'єкт на карті;
- буде або не буде використовуватися форма просторового об'єкта під час проведення географічного аналізу;
- просторовий об'єкт являє собою дані, до яких можна звертатися та візуалізувати через відношення з іншим просторовим об'єктом. Наприклад, до інформації про права на земельну ділянку можна звернутися, якщо вибрати земельну ділянку;
- просторовий об'єкт буде мати різні зображення у різних масштабах карти;
- текстові атрибути просторового об'єкта будуть відображені на картографічних продуктах або на екрані комп’ютера.

На цьому кроці для привласнення типу просторовим об'єктам користувач здійснює каталогізацію інформації, яка буде частиною словника просторових об'єктів.

Так, точка відображає місцеположення просторового об'єкта, форма якого дуже мала, щоб її визначити як область на карті даного масштабу, а лінія відображає місцеположення просторового об'єкта, форма якого дуже вузька, щоб її визначити як область на карті даного масштабу.

Область являє собою місцеположення та полігональну форму просторового об'єкта на карті даного масштабу, поверхня – форму просторового об'єкта так, як і область, а також форму, що виникає в результаті відмінностей значень висоти. Растр зображує область, але з використанням прямокутних чарунок (космічний знімок, аерофотознімок, безперервні дані), і може використовуватися для їх аналізу.

Примітиви подано у вигляді зображень, фотографій, креслень; вони являють собою цифрову картину, але такі подання не можна використовувати для аналізу.

Об'єкт являє собою просторовий об'єкт, який не слід відображати у вигляді точки, лінії або області, він немає ніякого геометричного або графічного подання.

Якщо під час проектування БГД виявляється, що просторові об'єкти можна подати у двох формах залежно від масштабу, то у словнику даних проекту вказуються обидві можливості і це враховується у процесі подальшого аналізу.

На другому кроці здійснюється просторове зображення об’єктів у векторному, растровому вигляді або за допомогою TIN. При цьому встановлюється просторове подання кожного типу даних. Для розглянутого вище прикладу просторове подання кожного типу даних можна навести у вигляді таблиці, наприклад, як таблиця 18.2.

Таблиця 18.2 – Дані, необхідні для забезпечення функцій організації

Записи про землю	
Тип даних	Просторове подання
Земельна ділянка	Область
Опис ділянки	Текст
Фотознімок ділянки	Зображення
Власник ділянки	Непросторовий об’єкт
Адреса	Місцеположення

Під час проектування БГД окремі просторові об'єкти доцільно зображувати за допомогою точок, ліній, областей, їх зручно наносити на карту, і вони є постійними. Так можна моделювати різні просторові об'єкти у векторному вигляді. При цьому, як зазначалося вище, точка є дуже малою, щоб зобразити її лінією або областю. Лінія є дуже вузькою, щоб

відобразити її областю. Область має довжину і ширину у масштабі карти, наприклад, земельна ділянка. Анотація є описовим підписом. Непросторовий об'єкт є негеографічним компонентом, наприклад, власник ділянки. Приклад векторного подання просторових об’єктів показано на рисунку 18.3.

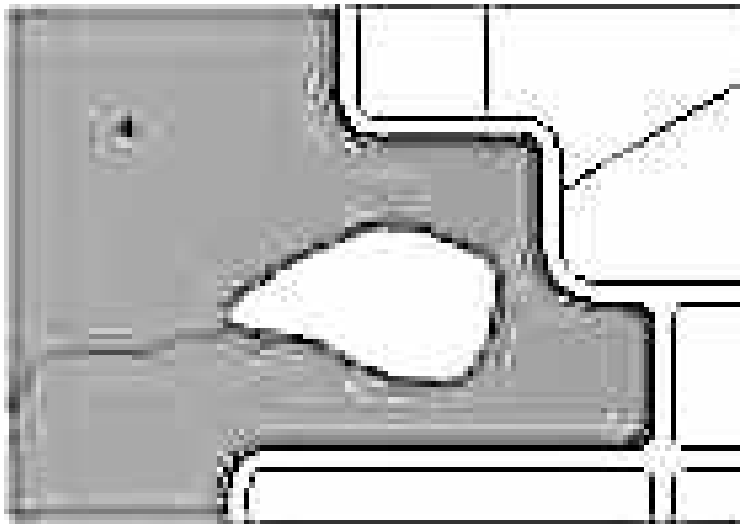


Рисунок 18.3 – Приклад векторного подання просторових об’єктів

Під час проектування БГД безперервні явища доцільно подавати за допомогою зображень, які мають різноманітне застосування в ГІС. Розробники БГД застосовують зображення для подання космічних та аерознімків, фотографій споруд і будь-яких документів, які були отримані за допомогою сканера. Таким чином, зображення у БГД являє собою файл, який містить безперервне відображення значень, аерофотознімок, копію плану або картинку будівлі. Приклад растрового подання просторових об’єктів показано на рисунку 18.4.

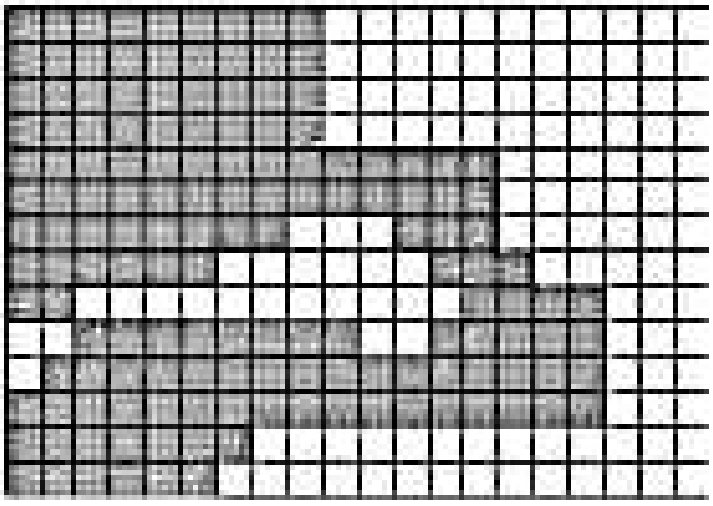


Рисунок 18.4 – Приклад растрового подання просторових об’єктів

Під час проектування БГД рельєф доцільно моделювати за допомогою поверхонь. При цьому моделюється безперервне явище, яке має значення висоти **z**. Під час моделювання поверхонь кращі результати можуть бути досягнуті, якщо використовувати нерегулярну тріангуляційну мережу (TIN). Така мережа створює систему точок або місцеположень зі значеннями висоти, яка утворює сітку для математичної апроксимації форми поверхні землі.

Приклад подання просторових об'єктів за допомогою TIN показано на рисунку 18.5.

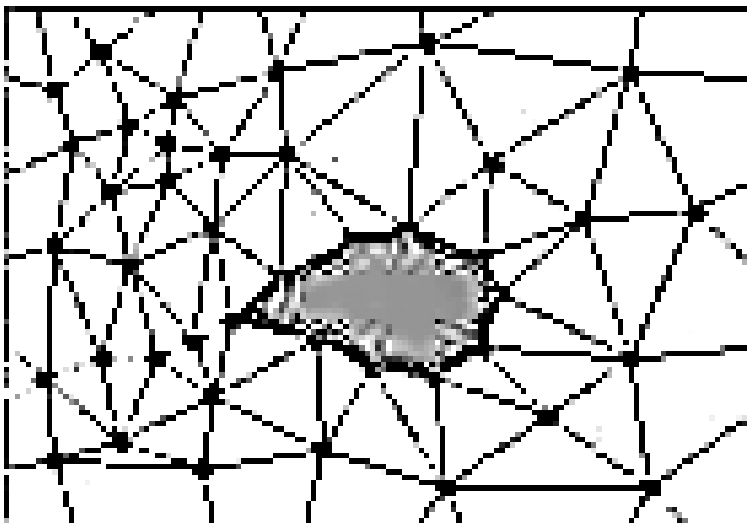


Рисунок 18.5 – Приклад подання просторових об'єктів за допомогою TIN

18.4 Узгодження з моделлю даних бази геоданих

Основною метою четвертого етапу є визначення того, які дані мають бути подано у конкретній базі геоданих, наприклад, у базі геоданих ArcInfo. При цьому для кожного з просторових типів, що були визначені на попередньому етапі, призначається відповідне подання у БГД, наприклад, у БГД ArcInfo. На цьому етапі розробники БГД переходять від визначення і обліку вимог користувача до розроблення ефективної схеми бази геоданих. Для реалізації цього етапу необхідно, щоб у команді були фахівці, які є обізнаними у моделі даних бази геоданих і можливостях аналізу, а також в інших технологіях організації даних у базі геоданих.

На цьому етапі розробники БГД виконують такі кроки:

- визначають відповідне подання бази геоданих для примітивів;
- забезпечують підтримку класів складних просторових об'єктів.

На першому кроці визначають подання даних у базі геоданих. Наприклад, БГД ArcInfo дозволяє зберігати дискретні примітиви як прості просторові об'єкти, складні просторові об'єкти та непросторові об'єкти.

Основним простим просторовим типом є точка. Наприклад, окрема точка, яка описує тип «історичний пам'ятник», є точковим просторовим об'єктом у БГД ArcInfo.

Просторові об’єкти, що пов'язані з вуличними сегментами, наприклад, типу «перехрестя», описуються за допомогою зв'язаної точки. При цьому просторовий об'єкт подається у БГД ArcInfo простим з'єднанням. Просторові об’єкти, що мають внутрішню топологію, наприклад, типу «пристрій очищення води», також можна описувати за допомогою зв'язаної точки, але у цьому випадку просторовий об'єкт у БГД ArcInfo подається складним з'єднанням. Ще одним просторовим типом є лінія. Наприклад, окрема лінія, яка описує тип «огорожа», зображується у БГД ArcInfo просторовим об'єктом «лінія».

Лінійний просторовий об'єкт, який застосовується у системі типу «дорожня мережа», подається у БГД ArcInfo просторовим об'єктом «просте ребро», а лінійний просторовий об'єкт зі з’єднаними секціями, який застосовується у системі типу «ділянки експлуатаційної лінії», – просторовим об'єктом «складне ребро».

Більш складним просторовим типом є область. Наприклад, окрема область, яка описує тип «парк», відображається у БГД ArcInfo полігональним просторовим об'єктом.

Області, що розподілені по поверхні і описують тип «рослинний покрив», подаються у БГД ArcInfo полігональним просторовим об'єктом, якому пізніше може бути надана плоска топологія.

Якщо просторовим типом є зображення (фотографія, відсканована карта, космічний знімок та ін.), то воно подається у БГД ArcInfo растром. Якщо просторовим типом є поверхня, для якої важливими є деталі рельєфу, то вона зображується у БГД ArcInfo нерегулярною тріангуляційною мережею (TIN). Поверхні, що охоплюють великі області, а також існуючі цифрові моделі рельєфу, подаються у БГД ArcInfo растром.

Непросторові об'єкти є у БГД ArcInfo об'єктами. До них належать примітиви, які не мають прямого географічного відображення, але пов'язані з географічними просторовими об'єктами.

На другому кроці розробники забезпечують підтримку класів складних просторових об'єктів.

Складні лінійні системи, що описують такі типи, як «транспорт» або «інженерні комунікації», подають у БГД ArcInfo геометричною мережею. Під час побудови геометричної мережі використовують спеціалізований метод, що спрощує її редагування.

Складні просторові області, що описують такі типи, як «системи земель» або «адміністративний розподіл», подають у БГД ArcInfo за допомогою плоскої топології, яка керує спільною геометрією набору класів об’єктів. Плоска топологія забезпечує те, що жоден просторовий об’єкт не може перетинати інший об’єкт без пересічення.

У розглянутому вище прикладі узгодження просторового подання даних з конкретною моделлю даних бази геоданих (БГД ArcInfo) можна навести у вигляді таблиці, наприклад, як таблиця 18.3.

Таблиця 18.3 – Узгодження просторового подання даних з базою геоданих ArcInfo

Записи про землю		
Тип даних	Просторове подання	Тип у БГД ArcInfo
Земельна ділянка	Область	Полігональний просторовий об’єкт
Опис ділянки	Текст	Просторовий об’єкт – підпис
Фотознімок ділянки	Зображення	Растр
Власник ділянки	Непросторовий об’єкт	Об’єкт
Адреса	Місцеположення	Адреса

18.5 Створення БГД у вигляді наборів географічних даних

На кінцевому (п’ятому) етапі проектування БГД проводиться ідентифікація та надаються назви наборам географічних даних, які містять різні примітиви, а у разі набору даних покриття створюються примітиви у покритті.

На цьому етапі розробники БГД виконують такі кроки:

- приписують примітиви до класів просторових об'єктів та підтипів;
- групують об’єднані набори класів об'єктів у геометричні мережі або плоскі топології;
- формують класи просторових об'єктів та набори даних у базі геоданих.

На першому кроці розробники БГД приписують примітиви до класів просторових об'єктів та підтипів. Необхідно зазначити, що до цього кроку вже були призначені типи просторових об'єктів та атрибути для примітивів. На цьому етапі визначають структуру класів просторових об'єктів з підтипами та встановлюють, чи є вони окремими класами просторових об'єктів або містяться у межах набору класів об'єктів.

Спочатку вирішують, чи має примітив бути відображеним у підтипі або у повному класі просторових об'єктів. При цьому розробник намагається консолідувати об’єднані примітиви як підтипи у межах класу просторових об'єктів. Таке об’єднання примітивів приводить до меншої кількості класів просторових об'єктів, що у свою чергу забезпечує кращу продуктивність баз геоданих.

Слід зазначити, що у деяких випадках замість об’єднання є необхідним створення нових класів просторових об'єктів. Такі ситуації виникають у такому разі:

- коли для подання кожної групи об’єднаних просторових об'єктів є необхідними різні спеціалізовані методи;

- коли набори атрибутів просторових об'єктів значно розрізняються;

- якщо є потрібними різні привілеї доступу до кожної групи просторових об'єктів;

- якщо деякі просторові об'єкти мають бути доступними через інші версії БГД, а деякі – ні.

На другому кроці групують об’єднані набори класів об'єктів у геометричні мережі або плоскі топології. При цьому визначаються топологічні ролі для класів просторових об'єктів.

Якщо тип просторового об'єкта – просте ребро, просте з'єднання, складне ребро або складне з'єднання, то його використовують у геометричній мережі. Всі класи просторових об'єктів у геометричній мережі мають бути розміщені у межах набору класів об'єктів. Це забезпечує те, що вони можуть спільно використовувати загальну просторову систему відліку.

Якщо тип примітивного просторового об'єкта – лінія або полігон, а примітив, охоплює повну область типу «земельна ділянка», то ці просторові об'єкти необхідно розміщувати у межах загального набору класів об'єктів. У межах загального набору класів об'єктів розташовуються також просторові об'єкти, якщо необхідно показати, що вони мають перерізи. У сучасних ГІС можна виконувати топологічне редагування класів просторових об'єктів. Таку множину називають плоскою топологією.

Якщо примітиви подано у вигляді простих просторових об'єктів,то їх можна розміщувати у межах набору класів об'єктів, який являє собою збирання довільної групи схожих класів просторових об'єктів.

Третій крок призначено для створення класів просторових об'єктів та наборів даних у базі геоданих. Цей крок починається одразу після визначення набору класів просторових об'єктів та їх топологічних асоціацій, при цьому вони групуються у базу геоданих.

Під час групування класів просторових об'єктів і наборів класів об'єктів в окремі бази геоданих розробники враховують таке:

- по-перше, якщо БГД створюють для великої організації, то різні структурні підрозділи використовують різні набори даних і працюють з ними. У зв’язку з цим структуру бази геоданих формують такою, щоб вона була близькою до структури організації;

- по-друге, в створюваній базі геоданих можна використовувати інші комерційні реляційні бази даних, але при цьому робота з кожною з них має бути організованою через окрему базу геоданих;

- по-третє, якщо створюють персональну базу геоданих, то можливими є практичні обмеження її розмірів, що у свою чергу може привести до тематичного або просторового поділення баз геоданих.

Для розглянутого вище прикладу структуру бази геоданих можна подати у вигляді таблиці, наприклад, як таблиця 18.4.

Таблиця 18.4 – Структура бази геоданих

Записи про землю			База геоданих
Тип даних	Просторове подання	Тип у БГД ArcInfo	Набір класів об'єктів
Земельна ділянка	Область	Полігональний просторовий об’єкт	Плоска топологія. Клас полігональних просторових об'єктів
Опис ділянки	Текст	Просторовий об’єкт – підпис	Клас просторових об'єктів – підписів
Фотознімок ділянки	Зображення	Растр	Набір растрових даних і растри
Власник ділянки	Непросторовий об’єкт	Об’єкт	Об'єктний клас. Клас відносин
Адреса	Місцеположення	Адреса	Покажчик та адреса

Таким чином, на першому етапі проектування БГД моделюють уявлення даних користувачем. При цьому з'ясовують організаційні функції, визначають дані, необхідні для підтримки цих функцій, а також формують дані у логічні групи.

На другому етапі визначають об'єкти і відношення. При цьому описують об'єкти і визначають відношення між ними, а також проводять документування моделі у вигляді схеми.

На третьому етапі розробники вибирають географічне подання. При цьому дискретні об'єкти зображають точками, лініями і областями, безперервні явища – растрами, а поверхні моделюють за допомогою TIN та растрів.

На четвертому етапі дані узгоджують з елементами бази геоданих. При цьому визначають тип геометрії дискретних просторових об'єктів і відношення між просторовими об'єктами, а також типи атрибутів для об'єктів.

На п’ятому етапі формують структуру бази геоданих. При цьому створюють системи просторових об'єктів і визначають їх топологічні зв'язки, задають координатні системи, а також визначають відношення і правила.

Запитання для самоперевірки

1. Як розробники визначають кожну з функцій організації ?
2. Для чого призначено діаграму UML, як її створюють та за допомогою якого програмного забезпечення ?
3. Як розробники вибирають географічне подання об’єктів ?
4. Які існують типи подання просторових об’єктів ?
5. Як розробники узгоджують подання даних з моделлю даних бази геоданих ?
6. Як розробники формують класи просторових об'єктів та набори даних у базі геоданих ?
7. Що слід враховувати під час групування класів просторових об'єктів та наборів класів об'єктів в окремі бази геоданих ?

Закінчення

Проектування бази геоданих по суті є таким самим, як і проектування будь-якої бази даних, але воно забезпечує моделювання (подання) географічних об’єктів реального світу. База геоданих об'єднує поведінку різних типів просторових об'єктів і типів ключових відношень між ними. Таке проектування реалізується за допомогою топології, яка в сукупності із інструментами і технологіями редагування бази геоданих дозволяє точніше моделювати реальні просторові відношення.

Традиційне проектування бази геоданих містить два основні етапи – з'ясування логічної моделі даних і фізичну реалізацію моделі бази геоданих. Так, за допомогою логічної моделі формується уявлення геоданих користувачем. Фізична модель бази геоданих будується на основі цієї моделі і у подальшому реалізується на основі технології реляційної бази даних. При цьому розробники використовують інструменти СУБД для задання схеми бази геоданих і створюють нову базу геоданих, в яку можна вводити географічні дані.

Завдяки об'єктно-орієнтованій структурі даних, яка відображає просторові і топологічні відношення географічних об'єктів, база геоданих одночасно розширює і спрощує процес проектування бази даних. Частиною цієї структури є спеціальний механізм для подання наборів об'єктів у вигляді інтегрованих систем, таких, як річкові або дорожні мережі, або набори земельних ділянок.

Об'єкти в базі геоданих зберігаються в пов’язаних реляційних таблицях. Частиною цих таблиць є зібрання просторових об'єктів. У інших таблицях задано відношення між просторовими об'єктами, наведено правила перевірки коректності і домени атрибутів. СУБД управляє цією структурою і забезпечує цілісність системи таблиць.

Сучасні бази геоданих розвиваються на основі мережних технологій, хмарних обчислень і методів штучного інтелекту, за допомогою яких зберігаються просторові інформаційні ресурси. У майбутньому бази геоданих можна буде застосовувати не тільки для дослідження інформації про Землю, а й для зберігання космічної інформації, наприклад, про малі небесні тіла. Це дозволить здійснювати системний аналіз просторової інформації, що зберігається в базі геоданих, з використанням методів космічної геодезії і космічної геоінформатики.

Бібліографічний список

1. Зейлер, М. Моделирование Нашего Мира : пособие ESRI® по проектированию баз геоданных / М. Зейлер. – Redlands : Environmental Systems Research Institute, 2004. – 255 с.

2. Грубер, М. Понимание SQL / М. Грубер. – М. : Дата+, 1993. – 291 с.

3. Харрис, М. Управление службами ArcSDE / М. Харрис. – М. : Дата+, 2003. – 130 с.

4. Маккой, Дж. Работа с базами геоданных: упражнения / Дж. Маккой. – М. : Дата+, 2003. – 208 с.

5. Грэй, П. Логика, алгебра и базы данных / П. Грэй. – М. : Машиностроение, 1989. – 368 с.

6. Бут, Б. Начало работы с ArcGis / Б. Бут, Э. Митчелл. – М. : Дата+, 2003. – 224 с.

7. Кузнецов, С. Д. Основы современных баз данных [Электронный ресурс] : Информ.-аналит. материалы / С. Д. Кузнецов. – М. : Центр информационных технологий. – Режим доступа: http://citforum.ru/database/osbd/contents.shtml.

8. Дейт, К. Введение в системы баз данных / К. Дейт. – М. ; СПб. : Вильямс. – 2000. – 848 с.

9. Геоинформатика / Е. Г. Капралов, А. В. Кошкарев, В. С. Тикунов и др.; под ред. В. С. Тикунова. – М. : Академия, 2005. – 480 с.

10. Шекхар, Ш. Основы пространственных баз данных / Ш. Шекхар, С. Чаула. – М. : КУДИЦ-ОБРАЗ, 2004. – 330 с.

11. Лурье, И. К. Геоинформационное картографирование. Методы геоинформатики и цифровой обработки космических снимков / И. К. Лурье. – М. : КДУ, 2008. – 424 с.

12. Диго, С. М. Базы данных, проектирование и использование / С. М. Диго. – М. : Финансы и статистика, 2005. – 124 с.

13. Гарев, А. Эффективная работа с СУБД / А. Гарев, Р. Ахаян, С. Макшарипов. – СПб. : Питер. – 2003. – 286 с.

14. Гринченко, Н. Н. Проектирование базы данных СУБД Microsoft Access / Н. Н. Гринченко. – М. : Горячая линия - Телеком, 2004. – 240 с.

Навчальне видання

Пащенко Руслан Едуардович

ПРОЕКТУВАННЯ БАЗ ГЕОДАНИХ

Редактор В. М. Коваль

Зв. план, 2018
Підписано до друку 23.07.2018
Формат 60х84 1/16. Папір офс. № 2. Офс. друк
Ум. друк. арк. 8,7. Обл.-вид. арк. 9,75. Наклад 50 пр.
Замовлення 266. Ціна вільна

Видавець і виготовлювач
Національний аерокосмічний університет ім. М. Є. Жуковського
«Харківський авіаційний інститут»
61070, Харків-70, вул. Чкалова, 17
http:// www.khai.edu
Видавничий центр «ХАІ»
61070, Харків-70, вул. Чкалова, 17
izdat@khai.edu

Свідоцтво про внесення суб'єкта видавничої справи
до Державного реєстру видавців, виготовлювачів і розповсюджувачів
видавничої продукції сер. ДК № 391 від 30.03.2001