Навчально-методична література

Міністерство освіти і науки України
Тернопільський національний технічний університет
ім. Івана Пулюя
Кафедра комп'ютерно-інтегрованих технологій

Конспект лекцій

з дисципліни

«Системи управління базами даних»

напряму підготовки 151«Автоматизація та комп'ютерно-інтегровані технології»

Тернопіль – 2022

УДК 681.3
ББК 32.973

Укладачі:
Чихіра І.В., канд. техн. наук, доцент,
Дідич І.С., Ph.D, асистент

Рецензент
Коноваленко І.В., канд. техн. наук, доцент

Схвалено та рекомендовано до друку на засіданні кафедри протокол №1 від 29.08.2022р.

Засідання факультету протокол №1 від 30.08.2022р.

Відповідальний за випуск *Чихіра І.В*., канд. техн. наук, доцент,

Лекція 1.

Вступ

1.1 Основні поняття

Організований певним чином масив даних, збережений в обчислювальній системі, називається базою даних.

Основні ідеї сучасної інформаційної технології базуються на концепції, відповідно до якої дані повинні бути організовані в бази даних з метою адекватного відображення реального миру, що змінюється, і задоволення інформаційних потреб користувачів. Ці бази даних створюються й функціонують під управлінням спеціальних програмних комплексів, називаних системами управління базами даних (СУБД).

Збільшення обсягу й структурної складності збережених даних, розширення кола користувачів інформаційних систем привели до широкого поширення найбільш зручних і порівняно простих для розуміння реляційних (табличних) СУБД. Для забезпечення одночасного доступу до даних безлічі користувачів, нерідко розташованих досить далеко друг від друга й від місця зберігання баз даних, створені мережні мультікористувальські версії БД заснованих на реляційній структурі. У них тим або іншим шляхом вирішуються специфічні проблеми паралельних процесів, цілісності (вірності) і безпеки даних, а також санкціонування доступу.

Переваги використання баз даних, пов’язані із централізованим управлінням за допомогою систем управління базами даних (СУБД).

Експертна система (ЕС) розглядається як система штучного інтелекту. Вона включає базу знань з набором правил, які дають змогу на основі фактів, отриманих користувачем, розпізнати ситуацію, здійснити діагностику та сформувати рішення або рекомендації [4, 6].

Система підтримки прийняття рішень (СППР) являє собою кінцевий продукт реалізації інформаційних технологій. Це — інтерактивна комп'ютерна система, що забезпечує діалог з користувачем й інтегрує моделі функціонування об'єкта для прийняття змістовного рішення. Як організований набір алгоритмів, оптимізаційних та імітаційних моделей, вона у сукупності з базами знань і базами даних забезпечує інтерпретацію та використання результатів вибору того чи іншого сценарію рішень [7]. При поєднанні СППР, побудованих на базі ЕС, формуються просторові системи підтримки прийняття рішень (ПСППР).

В основу формування будь-яких інформаційних баз покладено моделі організації даних для різних рівнів функціональних завдань.

Ядром будь-якої бази даних є модель даних. Модель даних являє собою безліч структур даних, обмежень цілісності й операцій маніпулювання даними. За допомогою моделі даних можуть бути представлені об'єкти предметної області й взаємозв'язку між ними.

1.2 Моделі даних

Модель даних — сукупність структур даних і операцій їхньої обробки. СУБД ґрунтується на використанні ієрархічної, мережний або реляційної моделі, на комбінації цих моделей або на деякій їхній підмножині [1]. Розглянемо три основних типи моделей даних: ієрархічну, мережну й реляційну.

Ієрархічна модель даних. Ієрархічна структура представляє сукупність елементів, зв'язаних між собою за певними правилами. Об'єкти, зв'язані ієрархічними відносинами, утворять орієнтований граф (перевернене дерево).

До основних понять ієрархічної структури ставляться: рівень, елемент (вузол), зв'язок. **Вузол** — це сукупність атрибутів даних, що описують деякий об'єкт. На схемі ієрархічного дерева вузли представляються вершинами графа. Кожний вузол на більше низькому рівні зв'язаний тільки одним вузлом, що перебуває на більше високому рівні. Ієрархічне дерево має тільки одну вершину (корінь дерева), не підлеглу ніякій іншій вершині й находящуюся на самому верхньому

(першому) рівні. Залежні (підлеглі) вузли перебувають на другому, третьому й т.д. рівнях. Кількість дерев у базі даних визначається числом кореневих записів.

До кожного запису бази даних існує тільки один (ієрархічний) шлях від кореневого запису.

Мережна модель даних. У мережній структурі при тих же основних поняттях (рівень, вузол, зв'язок) кожний елемент може бути пов'язаний з будь-яким іншим елементом.

Реляційна модель даних. Поняття реляційний (англ. relation — відношення) пов'язане з розробками відомого американського фахівця в області систем баз даних Е. Кодда [].

Ці моделі характеризуються простотою структури даних, зручним для користувача табличним поданням і можливістю використання формального апарата алгебри відносин і реляційного вирахування для обробки даних.

Реляційна модель орієнтована на організацію даних у вигляді двовимірних таблиць. Кожна **реляційна таблиця** являє собою двовимірний масив і має наступні властивості:

кожний елемент таблиці - один елемент даних; всі стовпці в таблиці однорідні, тобто всі елементи в стовпці мають однаковий тип (числовий, символьний і т.д.) і довжину; кожний стовпець має унікальне ім'я; однакові рядки в таблиці відсутні; порядок проходження рядків і стовпців може бути довільним.

Відносини представлені у вигляді **таблиць,** рядки яких відповідають кортежам або **записам,** а стовпці — атрибутам відносин, доменам, **полям.**

Поле, кожне значення якого однозначно визначає відповідний запис, називається **простим ключем** (ключовим полем). Якщо записи однозначно

визначаються значеннями декількох полів, то така таблиця бази даних має **складений ключ**.

Щоб зв' язати дві реляційні таблиці, необхідно ключ першої таблиці увести до складу ключа другої таблиці (можливий збіг ключів); у противному випадку потрібно ввести в структуру першої таблиці **зовнішній ключ** — ключ другої таблиці.

Розрізняють концептуальний, внутрішній і зовнішній рівні подання даних баз даних, яким відповідають моделі аналогічного призначення,

Концептуальний рівень відповідає логічному аспекту подання даних предметної області в інтегрованому виді. **Концептуальна модель** складається з безлічі екземплярів різних типів даних, структурованих відповідно до вимог СУБД до логічної структури бази даних.

Внутрішній рівень відображає необхідну організацію даних у середовищі зберігання й відповідає фізичному аспекту подання даних. **Внутрішня модель** складається з окремих екземплярів записів, фізично збережених у зовнішніх носіях.

Зовнішній рівень підтримує приватні подання даних, необхідні конкретним користувачам. **Зовнішня модель** є підмножиною концептуальної моделі. Можливе перетинання зовнішніх моделей за даними. Приватна логічна структура даних для окремого додатка (завдання) або користувача відповідає зовнішній моделі або підсхемі БД. За допомогою зовнішніх моделей підтримується санкціонований доступ до даних БД додатків (обмежені состав і структура даних концептуальної моделі БД доступних у додатку, а також задані допустимі режими обробки цих даних: введення, редагування, видалення, пошук).

Поява нових або зміна інформаційних потреб існуючих додатків вимагають визначення для них коректних зовнішніх моделей, при цьому на рівні концептуальної й внутрішньої моделі даних змін не відбувається. Зміни в концептуальній моделі, викликані появою нових видів даних або зміною їх структур, можуть зачіпати не всі додатки, тобто забезпечується певна незалежність програм від даних. Зміни в концептуальній моделі повинні відбиватися й у внутрішньої моделі, і при незмінній концептуальній моделі можлива самостійна модифікація внутрішньої моделі БД із метою поліпшення її характеристик (час доступу до даних, витрати пам'яті зовнішніх пристроїв і ін.). Таким чином, БД реалізує принцип відносної незалежності логічної й фізичної організації даних.

Проектування бази даних полягає в побудові комплексу взаємозалежних моделей даних.

Найважливішим етапом проектування бази даних є розробка інфологічної (інформаційно-логічної) моделі предметної області, не орієнтованої на СУБД. В інфологічній моделі засобами структур даних в інтегрованому виді відбивають состав і структуру даних, а також інформаційні потреби додатків (завдань і запитів).

Інформаційно-логічна (міфологічна) модель предметної області відбиває предметну область у вигляді сукупності інформаційних об'єктів і

їхніх структурних зв'язків.

Інфологічна модель предметної області будується першою. Попередня інфологічна модель будується ще на передпроектній стадії й потім уточнюється на більше пізніх стадіях проектування баз даних. Потім на її основі будуються концептуальна (логічна), внутрішня (фізична) і зовнішня моделі.

1.3 Особливості експертних систем для обробки даних

Структура експертної системи визначається наступними модулями:

1. Тимчасові бази даних, призначені для зберігання вихідних і проміжних даних поточного завдання;
2. Бази знань, призначені для зберігання довгострокових відомостей /фактів/ і правил маніпулювання даними;
3. Вирішатель /база програм/, що реалізує послідовність правил для рішення конкретного завдання на основі інформації, що зберігається в базах знань і базах даних;
4. Компонент придбання знань, що автоматизує процес наповнення бази знань;
5. Пояснювальний компонент, що формує пояснення про те, як система вирішувала поставлене завдання.

Експертні системи для обробки даних прийнято називати **статистичними експертними системами**. Такі системи за рахунок спільного користувальницького інтерфейсу повинні мати можливість допомогти починаючому користувачеві не тільки ввести результати спостережень, але й уточнити завдання обробки й, при необхідності, спланувати експеримент, що

дозволяє вирішити поставлене завдання. У базі знань експертної системи повинне зберігатися досить велика й постійно поповнювана кількість відомостей і правил, для того щоб забезпечити можливість рішення різноманітних завдань обробки даних. Пояснення про тім як система вирішувала поставлене завдання повинні бути зрозумілі фахівцеві в предметній області й, у теж самий час, містити досить інформації для аналізу вірогідності результатів обробки фахівцем з математичної статистики. Розробка експертних систем, призначених для обробки даних, натрапляє на величезні труднощі. «Інтелектуалізація» комп'ютерної обробки первинної інформації про навколишнє середовище ґрунтується, з одного боку, на ідеях і методах конкретної області знання, для якої створюється система обробки даних. З іншого боку, у комп'ютерній системі обробки використовуються різноманітні методи прикладної математики - математичної статистики, теорії рішення зворотних завдань і т.п. Відповідно, при створенні експертних систем обробки даних, з одного боку, доводиться враховувати методичні й метрологічні особливості методик виконання виміру, а з іншого боку - апріорні припущення й обмеження математичних алгоритмів обробки, що допускає участь досить великого колективу професіоналів - фахівців у предметній області,

математиків, програмістів і фахівців з розробки експертних систем і високу вартість розробки. Тому при наявності величезного числа систем загального призначення - пакетів для статистичної обробки даних, електронних таблиць і т.п., існує невелике число експертних систем, здатних автоматично провести весь цикл аналізу даних [3].

БД- системи забезпечують створення нових програмних модулів, які розширюють можливості ПІК та урізноманітнюють ступінь деталізації інформації відповідно до завдань, що вирішуються, та рівнів їхньої ієрархії. Особливістю БД-систем є наявність у їхньому складі специфічних методів аналізу просторової мінливості даних, які у сукупності із засобами вводу, зберігання, маніпулювання та представлення просторово координованої інформації (картографічної, атрибутивної) становлять основу БД-технологій. БД включає чотири основні підсистеми:

- збору даних, що вибирає, формує та здійснює попередню обробку інформації із різних джерел; ця підсистема також відповідає за

перетворення різних типів просторової інформації (її збір та складання карт);

• збереження та вибірки даних, що забезпечує організацію просторової інформації з метою її компонування, оновлення та редагування відповідно до конкретних запитів користувача;

• маніпулювання даними, їхніх аналізу та оцінки, що вирішує алгоритми різних завдань на основі цієї інформації, згруповує та розділяє її, установлює параметри та обмеження, виконує функції моделювання;

• виведення, що відображає всю інформаційну базу даних, або частину її в табличній, графічній (діаграмній), картографічній формах залежно від функціональних завдань БД.

Інформацію до бази подають у вигляді тематично (або предметно) та системно організованих даних.

Організація тематичних відомостей передбачає формування наборів галузевих даних, що характеризують, насамперед, природне середовище як природно-ресурсну основу об'єктів і виробничо-технологічну структуру території з виділенням базових і функціональних модулів.

Системна диференціація тематичних даних реалізується організацією спеціалізованих модулів **бази знань** (БЗ) і **бази даних** (БД), які становлять основу інформаційної бази ПІК об'єкту. Це, насамперед, блоки інформаційно-довідкової та нормативно-регламентуючої систем, локальної інформаційно-довідкової бази об'єкта досліджень, спеціалізованого тематичного картографічного фонду, банків даних оперативної, довгострокової та результуючої інформації.

За цільовим призначенням СППР ПІК об'єкту інформація в рамках бази знань і бази даних має низку певних особливостей і формується за суто адресними вимогами структурної організації даних в експертних і БД-системах.

Наповнення **бази знань** відбувається на основі узагальнення та систематизації досвіду досліджень з оцінювання природно-агромеліоративних об'єктів, організації контролю за їхнім станом, водогосподарських умов та ін.

База даних має забезпечити систему підтримки рішень базовою, довгостроковою, оперативною, а також результуючою інформацією для вибору сценаріїв та рекомендацій. Бази даних БД акумулюють інформацію у вигляді відповідним чином закодованих шарів однорідних картографічних даних і просторово прив'язаної до конкретної території або точки спостереження атрибутивної інформації.

Накопичення матеріалів у базах даних здійснюється покомпонентно відповідно до програм моніторингу, спеціальних обстежень та ви-пробувань, що підпорядковані різним організаціям з наступною адаптацією інформації до завдань ПІК об'єкту.

База даних формується на рівні об'єкта досліджень (регіональний, локальний чи детальний) з урахуванням певних вимог до наповнення залежно від показників, що фіксуються, та методів 'їхнього одержання.

До складу БД входить локальна інформаційно-довідкова база об'єкта досліджень (регіональний та локальний рівень узагальнення інформації), спеціалізований тематичний картографічний фонд (банки даних картографічної інформації) та банк відповідних атрибутів до карт, банки даних точкової інформації — оперативної, довгострокової і результуючої.

Наповнення бази знань та бази даних відбувається у декілька етапів. На першому етапі виконується упорядкування наявної, одержаної з різних джерел, інформації щодо території, у часі, за конкретним змістом, тематикою тощо. Другий етап включає узгодження упорядкованої інформації та її оптимізацію. На третьому етапі інформація інтегрується — вирішуються завдання більш високих рівнів складності та комплексності, тобто завдання з вибору і прийняття рішень.

Інформаційні бази мають залишатися відкритими, здатними до трансформації, забезпечуючи легкість модифікації системи як в еволюційному плані, так і для вирішення нових завдань, супроводжуватись гнучкими СУБД, системою SQL-запитань та розвинутим інтерфейсом.

СУБД являє собою сукупність програмних і технічних засобів, що забезпечують функціонування БД- та експертних систем, а саме — введення

інформації, її накопичення й оновлення, засобів збереження, пошуку та перетворення інформації, видачі матеріалів користувачу.

Відношення між окремими підсистемами, модулями та блоками інформаційної бази встановлюють формуванням файлів зв'язку (система SQL-запитань). При цьому стійкі відношення між об'єктами у підсистемах фіксуються сукупністю ієрархічних класифікаційних графів природних, природно-господарських об'єктів, адміністративно-територіальних та природно-територіальних одиниць, а підсистеми предметних даних вміщують файли, що описують у вигляді таблиць-відношень самі об'єкти та їхній стан.

Вимоги до інформаційної бази ПІК об'єкту можуть змінюватись, розширюватись або звужуватись залежно від конкретного призначення БД та ЕС, складу завдань, що вирішуються, та особливостей об'єкта досліджень [8].

1.4. Поняття бази даних

Мета будь-якої інформаційної системи — обробка даних про об'єкти реального миру. У широкому змісті слова база даних — це сукупність відомостей про конкретні об'єкти реального миру в якій-небудь предметній області. Під **предметною областю** прийнято розуміти частину реального миру, що підлягає вивченню для організації управління й в остаточному підсумку автоматизації, наприклад, підприємство, вуз і т буд.

Створюючи базу даних , користувач прагне впорядкувати інформацію з різними ознаками і швидко робити вибірку з довільним сполученням ознак. Зробити це можливо, тільки якщо дані структуровані.

Структурування — це введення угод про способи подання даних.

Неструктурованими називають дані, записані, наприклад, у текстовому файлі.

Користувачами бази даних можуть бути різні прикладні програми, програмні комплекси, а також фахівці предметної області, що виступають у ролі споживачів або джерел даних, називані **кінцевими користувачами**.

У сучасній технології баз даних передбачається, що створення бази даних, її підтримка й забезпечення доступу користувачів до неї здійснюються

централізовано за допомогою спеціального програмного інструментарію — системи управління базами даних.

База даних (БД) — це пойменована сукупність структурованих даних, що ставляться до певної предметної області.

Система управління базами даних (СУБД) — це комплекс програмних і мовних засобів, необхідних для створення баз даних, підтримки їх в актуальному стані й організації пошуку в них необхідної інформації.

Централізований характер управління даними в базі даних допускає необхідність існування деякої особи (групи осіб), на яку покладають функції адміністрування даними, збереженими в базі.

За технологією обробки дані бази даних підрозділяються на централізовані й розподілені.

Централізована база даних зберігається в пам'яті однієї обчислювальної системи. Якщо ця обчислювальна система є компонентом мережі ЕОМ, можливий розподілений доступ до такої бази. Такий спосіб використання баз даних часто застосовують у локальних мережах ПК.

Розподілена база даних складається з декількох, можливо пересічних або навіть дублюючих одну одну частин, збережених у різних ЕОМ обчислювальній мережі. Робота з такою базою здійснюється за допомогою системи управління розподіленою базою даних (СУРБД).

За способом доступу до даних бази даних розділяються на бази даних з локальним доступом і бази даних з вилученим (мережним) доступом.

Система баз даних (database system) — це, по суті, не що інше, як комп'ютеризована система зберігання записів. Користувачеві цієї системи надається можливість виконувати безліч різних операцій над такими файлами, наприклад:

- додавати нові порожні файли в базу даних;
- додавати нові дані в існуючі файли;
- вести пошук даних в існуючих файлах;
- змінювати дані в існуючих файлах;
- видаляти дані з існуючих файлів;

• видаляти існуючі файли з бази даних, тобто позбуватися від їхнього вмісту.

Система баз даних - це комп'ютеризована система зберігання записів; тобто це комп'ютеризована система, основна мета якої – зберігати інформацію й надавати її на вимогу. До інформації може відноситися все, що заслуговує на увагу окремого користувача або підприємства, що використовує систему.

Однокористувальницька система (single-user system) — це система, у якій у те саме час до бази даних може одержати доступ не більше одного користувача; багатокористувальницька система (multi-user system) — це система, у якій до бази даних можуть одержати доступ відразу кілька користувачів.

У загальному випадку дані в базі даних (принаймні в більших системах) є інтегрованими й загальними. Ці два аспекти, інтеграція й дозвіл загального доступу, являють собою найбільш важливу перевагу використання систем баз даних на "великому" устаткуванні; і щонайменше один з них - інтеграція - є перевагою їхнього використання на "малому" устаткуванні.

• Під поняттям інтегровані дані мається на увазі можливість представити базу даних як об'єднання декількох окремих файлів даних, повністю або частково, що перекриваються.

Під поняттям загальні дані мається на увазі можливість використання окремих областей даних у базі даних декількома різними користувачами, тобто кожний із цих користувачів може мати доступ до однієї й тій же області даних (причому різні користувачі можуть використовувати ці дані для різних цілей). Як уже згадувалося, різні користувачі можуть мати доступ навіть до однієї й тій же області даних у той саме час (одночасний доступ) [10].

у Вхідні дані — це інформація, передана системі (звичайно з термінала або робочої станції). Така інформація може стати причиною змін постійних даних (вона може стати частиною постійних даних), але не є частиною бази даних як такої.

• Вихідні дані — це повідомлення й результати, видавані системою (звичайно видаються на печатку або відображаються на екрані). І знову ж цю інформацію можна брати з постійних даних, але її не можна розглядати як частину бази даних.

Розходження між постійними й транзитними даними не можна назвати чітким - воно в деякій мері залежить від контексту (наприклад, від того, як використовуються дані). Однак допускаючи, що це розходження доступно принаймні на інтуїтивному рівні, можна дати більше точне визначення терміна "база даних":

Переваги системи баз даних у порівнянні із традиційним паперовим методом змісту записів наступні.

- Компактність. Немає необхідності в багатотомних паперових картотеках.
- Швидкість. Комп'ютер може вести пошук і змінювати дані набагато швидше людини. Зокрема, на спеціальні питання, що виникають у процесі роботи (наприклад: "Яких труб в нас зараз більше – ВТ-12 або ВТ-15?"), можна одержати відповідь швидко, не затрачаючи часу на візуальний пошук.
- Низькі трудовитрати. Немає необхідності в стомлюючій ручній роботі над картотекою. Механічну роботу машини завжди виконують краще.
- Застосовність. Точна, свіжа інформація в будь-який момент під рукою.

Ці переваги здобувають ще більше значення в багатокористувальницькому середовищі, де база даних, імовірно, більше й складніше однокористувальницької. Крім того, багатокористувальницьке середовище має додаткову перевагу: система баз даних надає об'єкту централізоване управління його даними (а таке управління є найціннішою властивістю бази даних). Уявіть собі протилежну ситуацію – об'єкт, що не використовує систему баз даних: для кожного окремого додатка створюються свої файли, найчастіше розташовувані на окремих магнітних стрічках або дисках, у результаті чого дані виявляються розрізненими. Систематично управляти такими даними дуже складно[10].

1.5 Переваги централізованого підходу в управлінні даними

- Можливість скорочення надмірності.

У системах, що не використовують бази даних, кожний додаток має свої файли. Це часто приводить до надмірності збережених даних, а отже, до

марнотратства пам'яті. Наприклад, як додаток, пов'язане з викладачів, так і додаток, пов'язаний з обліком навчання підвищення кваліфікації викладачів, можуть мати власний файл із відомчою інформацією про викладачів. Як відзначено вище, ці два файли можна об'єднати з усуненням надмірності (однакової інформації) за умови, якщо адміністратор даних знає про те, які дані потрібні для кожного додатка, тобто якщо на об'єкті здійснюється необхідне загальне управління.

У цьому випадку не мається на увазі, що надмірність даних може або повинна бути повністю усунута. Іноді вагомі практичні або технічні причини

вимагають наявності декількох копій збережених даних. Однак така надмірність повинна строго контролюватися, тобто враховуватися в СУБД; також повинна бути передбачена можливість "множинного відновлення". Можливість усунення (певною мірою) суперечливості.

У дійсності цей наслідок попереднього пункту. Візьмемо приклад з життя — нехай викладач ЕЗ, що працює на кафедрі БД-технологій представлений двома різними записами в базі даних. Припустимо, що в СУБД не враховане це роздвоєння (тобто надмірність не контролюється). Тоді рано або пізно обов'язково виникне ситуація, при якій ці два записи перестануть погоджуватися а саме: одна з них буде змінена, а друга —ні. У цьому випадку база даних стане суперечлива. Ясно, що суперечлива база даних може видавати користувачеві неправильну, суперечливу інформацію.

Також очевидно, що якщо який-небудь факт представлений одним записом (тобто при відсутності надмірності), те протиріч виникнути не може. Протиріч можна також уникнути, якщо не видаляти надмірність, а контролювати її (передбачивши це відповідним чином у СУБД), тоді, з погляду користувача, база даних ніколи не буде суперечливою. Це забезпечується тим, що якщо відновлення вноситься в один запис, то воно автоматично повинне поширюватися на всі інші. Цей процес називається множинним відновленням, де під відновленням розуміються будь-які операції вставки, видалення або відновлення.

Можливість загального доступу до даних.

Загальний доступ до даних означає можливість доступу до них декількох існуючих додатків бази даних і розробки нових додатків для роботи із цими ж

даними. Інакше кажучи, нові додатки можуть одержати доступ до тих же даним, і при цьому немає необхідності в створенні нових даних.

Можливість дотримання стандартів.

Завдяки централізованому управлінню АБД (під управлінням адміністратора даних) може забезпечувати подання даних у певних стандартах . Стандарти можуть бути корпоративними, настановними, відомчими, промисловими, національними й інтернаціональними. Стандартизація подання даних найбільш важлива для обміну й перенесення даних між системами. Крім того, стандарти ідентифікації даних і документації важливі й для спільного використання даних, і для їхнього розуміння.

Можливість введення обмежень для забезпечення безпеки.

Завдяки повному контролю над базою даних АБД може забезпечити доступ до бази даних через певні канали, а отже, може визначити правила безпеки, які будуть перевірятися при спробі доступу до уразливих даних (знову ж під керівництвом адміністратора даних). Для різних типів доступу (вибірки, вставки, видалення й т.д.) і різних частин бази даних можна встановити різні правила.

Можливість забезпечення цілісності даних.

Завдання цілісності полягає в забезпеченні правильності й точності даних у базі даних . Протиріччя між двома записами, що представляють один "факт", являє приклад недоліку цілісності ; звичайно, ця проблема може виникнути лише при наявності надмірності в збережених даних. Але навіть якщо надмірність відсутня, база даних може містити неправильну інформацію. Централізоване управління базою даних дозволяє уникнути подібних проблем - наскільки їх взагалі можливо уникнути. Для цього адміністратор даних визначає (а АБД реалізує) правила цілісності, які застосовуються при будь-якій спробі проробити яку-небудь операцію відновлення.

Відзначимо також, що інтеграція даних для багатокористувальницьких систем баз даних навіть більше важлива, чим для "приватних файлів", саме тому, що до такої бази даних є загальний доступ. При відсутності належного контролю один користувач може некоректно обновити дані, і від цього постраждають інші ні в чому не винні користувачі. Варто також сказати, що в більшості продуктів баз

даних підтримка контролю цілісності розвинена слабко, хоча в цей час у цьому напрямку спостерігаються деякі поліпшення.

Можливість збалансувати суперечливі вимоги.

Знаючи загальні вимоги всього об'єкту (а не вимоги кожного окремого користувача), АБД (звичайно , під управлінням адміністратора даних) може структуризувати базу даних таким чином, щоб обслуговування в цілому для підприємства було найкращим.

Поняття бази даних тісно пов'язане з такими поняттями структурних елементів, як поле, запис, файл (таблиця).

Поле — елементарна одиниця логічної організації даних, що відповідає неподільній одиниці інформації — реквізиту. Для опису поля використовуються наступні характеристики:

ім'я, наприклад. Прізвище, Ім'я, По батькові, Дата народження;

тип, наприклад, символьний, числовий, календарний;

довжина, наприклад, 15 байт, причому буде визначатися максимально можливою кількістю символів;

точність для числових даних, наприклад два десяткових знаки для відображення дробової частини числа.

Запис — сукупність логічно зв'язаних полів. Екземпляр запису - окрема реалізація запису, що містить конкретні значення її полів.

Файл (таблиця) — сукупність екземплярів записів однієї структури.

У структурі запису файлу вказуються поля, значення яких є ключами первинними (ПК), які ідентифікують екземпляр запису, і вторинними (ВК), які виконують роль пошукових або групованих ознак (за значенням вторинного ключа можна знайти кілька записів) [10].

Контрольні запитання

1. Предметна область і бази даних.

2. Роль баз даних в процесі формування інформаційних ресурсів, в тому числі для цілей управління територіями.
3. Класифікація інформаційних систем.
4. Використання БД в гідромеліорації.
5. Експертні системи для обробки даних.

Лекція 2
МОДЕЛІ ДАНИХ

2.1 Види моделей даних

На великих ЕОМ сімдесятих років програмісти прагнули з максимальною ефективністю використовувати машинний час і пам'ять. Тому широке поширення одержав ієрархічний підхід до організації баз даних. **Ієрархічні бази даних** організовуються у вигляді дерев, що припускає нерівноправність між даними – одні дані виявляються жорстко підлеглі іншим. Така організація даних нагадує деякі схеми побудови баз знань в експертних системах і забезпечує високоефективний пошук інформації. Але ієрархічний підхід до організації баз даних має й очевидні недоліки, наприклад, необхідність жорстко визначати зв'язок між даними, що істотно ускладнює організацію інформації. Щоб перебороти подібні недоліки була запропонована **мережна модель даних**, у якій крім вертикальних зв'язків між даними, передбачалися й горизонтальні. Прикладом реалізації такої моделі може служити система директорій (фолдерів), що дозволяє організувати інформацію на жорсткому диску персонального комп'ютера. Але й ця модель вимагає досить більших зусиль при організації інформації.

Дані в базі даних називають " постійними" (хоча насправді вони можуть недовго залишатися такими!). Під словом "постійні" маються на увазі дані, які відрізняються від інших, більше мінливих даних, таких як проміжні результати, вхідні й вихідні дані, оператори управління робочі черги, програмні блоки управління й взагалі всі транзитні дані.

Поєднуючи частки подання про вміст бази даних , отримані в результаті опитування користувачів, і свої подання про дані, які можуть знадобитися в майбутніх додатках, АБД спочатку створює узагальнений неформальний опис створюваної бази даних. Це опис, виконаний з використанням природної мови, математичних формул, таблиць, графіків і інших засобів, зрозумілих всім людям, що працюють над проектуванням бази даних, називають інфологіченою моделлю даних.

Така людино-орієнтована модель повністю незалежна від фізичних параметрів середовища зберігання даних. Зрештою цим середовищем може бути пам'ять людини, а не ЕОМ. Тому інфологічна модель не повинна змінюватися доти, поки якісь зміни в реальному світі не зажадають зміни в ній деякого визначення, щоб ця модель продовжувала відбивати предметну область.

Інші моделі є комп'ютеро-орієнтованими. З їхньою допомогою СУБД дає можливість програмам і користувачам здійснювати доступ до збережених даних лише за їхними іменами, не піклуючись про фізичне розташування цих даних. Потрібні дані відшукуються СУБД на зовнішніх запам'ятовувальних пристроях по фізичній моделі даних.

Тому що зазначений доступ здійснюється за допомогою конкретної СУБД , то моделі повинні бути описані мовою опису даних цієї СУБД. Такий опис, створюваний АБД за інфологічною моделі даних, називають даталогічною моделлю даних.

Трьохрівнева архітектура (інфологічний, даталогічний і фізичний рівні) дозволяє забезпечити незалежність збережених даних від їхніх програм, що використовують. АБД може при необхідності переписати збережені дані на інші носії інформації й (або) реорганізувати їхню фізичну структуру, змінивши лише фізичну модель даних. АБД може підключити до системи будь-яке число нових користувачів (нових додатків), доповнивши, якщо треба, даталогічну модель. Зазначені зміни фізичної й даталогіної моделей не будуть замічені існуючими користувачами системи (виявляться "прозорими" для них), так само як не будуть замічені й нові користувачі. Отже, незалежність даних забезпечує можливість розвитку системи баз даних без руйнування існуючих додатків.

Інфологічна модель відображає реальний мир у деякі зрозумілі людині концепції, повністю незалежні від параметрів середовища зберігання даних. Існує безліч підходів до побудови таких моделей: графовые моделі, семантичні мережі, модель " сутність-зв'язок" і т.д. Найбільш популярної з них виявилася модель " сутність-зв'язок".

Інфологічна модель повинна бути відображена в комп'ютеро-орієнтовану даталогічну модель, "зрозумілу " СУБД. У процесі розвитку теорії й практичного

використання баз даних, а також засобів обчислювальної техніки створювалися СУБД, що підтримують різні даталогічні моделі.

Спочатку стали використовувати ієрархічні даталогічні моделі. Простота організації, наявність заздалегідь заданих зв'язків між сутностями, подібність із фізичними моделями даних дозволяли домагатися прийнятної продуктивності ієрархічних СУБД на повільних ЕОМ досить обмеженими обсягами пам'яті. Але, якщо дані не мали деревоподібної структури, то виникала маса складностей при побудові ієрархічної моделі й бажанні домогтися потрібної продуктивності.

Мережні моделі також створювалися для мало ресурсних ЕОМ. Це досить складні структури, що складаються з "наборів" - пойменованих дворівневих дерев. "Набори" з'єднуються за допомогою " записів-зв'язувань", створюючи ланцюжки й т.д. При розробці мережних моделей було вигадане безліч "маленьких хитростей", що дозволяють збільшити продуктивність СУБД, але істотно ускладнили останні. Прикладний програміст повинен знати масу термінів, вивчити кілька внутрішніх мов СУБД, детально представляти логічну структуру бази даних для здійснення навігації серед різних екземплярів, наборів, записів і т.п. Один з розроблювачів операційної системи UNIX сказав "Мережна база - це самий вірний спосіб втратити дані".

Складність практичного використання ієрархічних і мережних СУБД змушувала шукати інші способи подання даних. Наприкінці 60-х років з'явилися СУБД на основі інвертованих файлів , що відрізняються простотою організації й наявністю досить зручних мов маніпулювання даними. Однак такі СУБД володіють рядом обмежень на кількість файлів для зберігання даних, кількість зв'язків між ними, довжину запису й кількість її полів.

Фізична організація даних впливає на експлуатаційні характеристики БД. Розроблювачі СУБД намагаються створити найбільш продуктивні фізичні моделі даних, пропонуючи користувачам той або інший інструментарій для поднастройки моделі під конкретну БД. Розмаїтість способів коректування фізичних моделей сучасних промислових СУБД не дозволяє розглянути їх у цьому розділі.

2.2 Структури баз даних для управління даними

Організований набір взаємозалежних файлів даних називається базою даних. Складність роботи з множинними файлами в базі даних вимагає більш конкретного управління, реалізованого системою керування базою даних (СУБД). Хоча постійно створюються всі нові реалізації структур баз даних, існує всього три основних типи: ієрархічна (деревоподібна), мережна і реляційна (таблична) структури баз даних.

Ієрархічна структура даних

з багатьох випадках існує взаємозв'язок між даними, що має назву відношенням "один до багатьох" [P.A. Burrough, 1983]. Це відношення має на увазі, що кожен елемент даних має прямий зв'язок з деяким числом так званих "нащадків", і, звичайно кожен такий нащадок, у свою чергу, може мати зв'язок зі своїми нащадками і т.д. Як випливає з назви, предки і нащадки прямо зв'язані між собою, що робить доступ до даних простим і ефективним. Така система добре ілюструється ієрархічною системою класифікації рослин і тварин, називаною таксономією.

Якщо інформація про ключову ознаку недостатня, то не можна просуватися по дереву. У дійсності, природа ієрархічної системи вимагає явного визначення кожного, відносини для того, щоб створити саму структуру і її правила розгалуження. Головною перевагою такої системи є те, що в ній дуже легко шукати, оскільки вона добре визначена і може відносно легко розширюватися додаванням нових галузей і формулюванням нових правил розгалуження. Для створення ієрархічної структури зовсім необхідне знання всіх можливих питань, що можуть задаватися, оскільки ці питання використовуються як основа для розробки правил розгалуження або ключів.

Мережні структури. Можливості швидкого пошуку , виконуваного в ієрархічній структурі даних, визначаються структурою самого дерева. Атрибутивні і геометричні дані можуть зберігатися в різних місцях, що зажадає установлення великого числа зв'язків між графічною й атрибутивною частинами

БД. У такому випадку потенційне число розгалужень і зв'язаних з ними ключів ієрархічної структури може стати дуже великим. Мережні БД використовують відношення "багатьох до багатьох", при якому один елемент може мати багато атрибутів, при цьому кожен атрибут зв 'язаний явно з багатьма елементами. Для реалізації таких відносин разом з кожним елементом даних може бути зв'язана спеціальним перемінним, що має назву покажчик, що направляє до всіх інших елементів даних, зв'язаних з цим. Замість того, щоб обмежуватися деревоподібною структурою зв'язків, кожен окремий елемент даних може бути прямо зв'язаний з будь-яким місцем бази даних, без уведення відносини "предок-нащадок".

Мережні структури звичайно розглядаються як удосконалення ієрархічних структур , оскільки вони менш тверді і можуть представляти відношення "багато хто до багатьох". Тому вони допускають набагато велику гнучкість пошуку, ніж ієрархічні структури. Також на відміну від ієрархічних структур вони зменшують надмірність даних. Їхнім головним недоліком є те, що у великих БД кількість покажчиків може стати дуже великим, вимагаючи значної витрат пам'яті. Додатково, хоча зв'язки між елементами даних більш гнучкі, вони все-таки повинні бути явно визначені за допомогою покажчиків.

Реляційні бази даних

Недоліків великої кількості покажчиків можна уникнути використовуючи ще одну структуру баз даних - реляційну. В ній дані зберігаються як упорядковані записи або рядки значень атрибутів. Атрибути об'єктів групуються в окремих рядках у виді так званих відносин, оскільки вони зберігають свої положення в кожнім рядку і виразно зв'язані один з одним [Pr.G. Heal ey, 1991]. Кожний стовпчик містить значення одного атрибута для всього набору об'єктів.

Реляційні системи засновані на наборі математичних принципів, називаних реляційною алгеброю або алгеброю відносин [J.D. Ullman, 1982], що встановлює правила проектування і функціонування таких систем. Оскільки реляційна алгебра ґрунтується на теорії множин, кожна таблиця відносин функціонує як безліч, і перше правило говорить, що таблиця не може мати рядок, що цілком збігається з яким-небудь іншим рядком. Оскільки кожний з рядків унікальний,

одна або трохи колонок можуть використовуватися для визначення критерію пошуку, визначеного імені з першого стовпчика. Такий критерій пошуку називається первинним ключем для пошуку значень в інших колонках бази даних [11].

Реляційні системи коштовні тим, що дозволяють нам збирати дані в досить прості таблиці, при цьому задачі організації даних також прості. При необхідності ми можемо стикувати рядка з однієї таблиці з відповідними рядками з іншої таблиці , використовуючи сполучний механізм, називаний реляційним з'єднанням. Будь-яка кількість таблиць може бути "зв'язана". З'єднання відбувається по рівності значень стовпчика первинного ключа однієї таблиці з іншим стовпчиком другої таблиці. Стовпчик другої таблиці, з яким зв'язаний первинний ключ, називається зовнішнім ключем.

Для визначення виду, який таблиці повинні мати, встановлений набір правил, називають **нормальними формами** [E.F. Codd, 1970]. **Перша нормальна форма** затверджує, що таблиця повинна складатися з рядків і стовпчиків і, оскільки стовпчики будуть використовуватися як ключі пошуку, кожній з них на кожнім рядку повинне знаходитися тільки одне значення. **Друга нормальна форма** вимагає, щоб кожен стовпчик, що не є первинним ключем, цілком залежав від первинного ключа. Це спрощує таблиці і зменшує надмірність обмеженням, що кожен рядок даних може бути знайдений тільки через його первинний ключ. **Третя нормальна форма,** зв'язана з другою, вимагає, щоб стовпчики, що не є первинним ключем, "залежали" від первинного ключа, у той час, як первинний ключ не залежить від якого-небудь не первинного ключа. Правила нормальних форм були підсумовані Кентом [W. Kent, 1983]. Ці правила досить корисні і повинні строго виконуватися.

2.3 Багатошарові моделі даних БД

• той час, як растрові і векторні структури даних дають засоби відображення окремих просторових феноменів, все-таки існує необхідність розробки більш складних підходів, що мають назву моделі даних, для включення в базу даних взаємин об'єктів, зв'язування об'єктів і їхніх атрибутів, забезпечення спільного аналізу декількох шарів карти.

Растрові моделі. Кожен осередок у найпростішій такій структурі зв'язаний з одним значенням атрибута. Для створення растрової тематичної карти збирають дані про визначену тему у формі двомірного масиву осередків, де кожен осередок представляє атрибут окремої теми. Такий двомірний масив називається покриттям. Використовують покриття для представлення різних типів тематичних даних.

- той час як растрові БД традиційно розроблялися для представлення одиночних атрибутів, збережених індивідуально для кожного осередку растра, деякі з них достигли стану використання прямих зв'язків з існуючими СУБД. Такі розширення растрової моделі даних дозволили також установити прямий зв'язок з БД, що використовують векторну структуру графічних даних. Оскільки такі інтегровані растрово-векторні системи включають модулі, що перетворять інформацію з растрової форми у векторну і назад, користувач може використовувати достоїнства обох структур даних. Процес перетворення часто прозорий, так що користувачеві навіть не потрібно турбуватися про вихідну структуру даних.

Векторні моделі даних. Векторні структури даних дають представлення більш інтуїтивно зрозумілим способом і очевидно більше нагадують добре відомі паперові карти. Існують кілька способів об'єднання векторних структур даних у векторну модель даних, що дозволяє досліджувати взаємозв'язки між показниками всередині одного покриття або між різними покриттями. Найпростішою векторною структурою даних є ланцюгові-модель [I. Dangermond, 1982], що по суті переводить "один в один" графічне зображення карти.

В цій моделі сусідні області повинні мати різні ланцюжки для загальних сторін. Тобто, не існує областей, для яких який-небудь ланцюжок був би загальним. Кожна сторона кожної області має свій унікальний набір ліній і пару координат. На відміну від ланцюгові-моделі, топологічні моделі [I. Dangermond, 1982], містять топологічну інформацію в явному виді. Для підтримки сучасних аналітичних методів потрібно ввести комп'ютер якнайбільше явної топологічної інформації. Топологічна модель даних поєднує рішення деяких з найбільш часто використовуваних у аналізі функцій. Це забезпечується включенням у структуру

даних інформації про суміжність для усунення необхідності визначення її при виконанні багатьох операцій. Топологічна інформація описується набором вузлів і дуг. Вузол (більше, ніж просто крапка, звичайно це перетинання двох або більш дуг, і його номер використовується для посилання на будь-яку дугу, якій він належить. Кожна дуга починається і закінчується або в крапці перетинання з іншою дугою, або у вузлі, що не належить іншим дугам. Дуги утворяться послідовностями відрізків, з'єднаних проміжними (формотворними) крапками. У цьому випадку кожна лінія має два набори чисел: пари координат проміжних крапок і номера вузлів. Крім того, кожна дуга має свій ідентифікаційний номер, що використовується для вказівки того, які вузли представляє її початок і кінець.

Неефективність збереження і пошуку, властивої базової топологічної моделі усувається роздільним збереженням кожного типу об'єктів (крапки, лінії, області). Ці окремі об'єкти потім зв'язуються в ієрархічну структуру даних, де крапки через покажчики зв'язані з лініями, а лінії - з областями. Кожен набір відрізків, називаний у даній моделі ланцюжком, починається і закінчується у визначених вузлах (перетинаннях двох ланцюжків).

Існування **гібридної моделі даних** БД - підтвердження того, що хоча її структури даних ефективні стосовно графічних характеристик об'єктів, їм не дістає тієї ж ефективності в управлінні атрибутивними даними [P. Ar onson, 1985; S.Morehouse, 1985]. І навпаки, СУБД загальновизнані як засіб управління атрибутивними типами даних, але погано пристосовані до роботи з графічними об'єктами. Виглядає цілком логічним, що програмне об'єднання цих двох технологій дозволить узяти краще з кожної. Для реалізації цього підходу координатні і топологічні дані, необхідні для графіки, зберігаються як окремий набір файлів. Таблиці атрибутів, що містять усі необхідні описові дані для кожного графічного об'єкта, зберігаються окремо або в інших файлах, або під управлінням СУБД загального призначення. Зв 'язок між графікою й атрибутами здійснюється через ідентифікаційні коди графічних об'єктів, що маються в графічних файлах, і які також зберігаються в окремому стовпчику атрибутивної БД. Завдяки можливості зовнішнього збереження багатьох атрибутів для кожного об'єкта ростуть аналітичні можливості і можлива економія пам'яті.

Іншим підходом до збереження графічних і атрибутивних даних є **інтегрована модель даних**. У цьому випадку БД є процесором просторових запитів, надстроєним над стандартної СУБД, що використовується для збереження як атрибутивної, так і графічної інформації [Gaptill, 1987; Morehous e, 1989]. Інтегрована система зберігає координати об'єктів карти атрибути в різних таблицях однієї БД, що зв'язуються механізмом, подібним реляционному з'єднанню [R.G. Healey, 1991]. Крім того, атрибути можуть розміщатися в тих же таблицях, що і графіка.

Існують два способи збереження координатної інформації в реляційних таблицях. У першому записуються окремі пари координат, що представляють крапкові об'єкти, а також кінцеві і проміжні крапки ліній і границь областей, як індивідуальні атоми, або рядка, бази даних. Інтегрована модель може записувати в один стовпчик таблиці цілі ланцюжки координатної інформації. Таким чином, одна область може бути описана одним рядком таблиці, що містить в одній колонці ідентифікатор області, а в іншій - список ідентифікаторів ліній. Тоді лінії, ідентифікуємі за цим кодом в окремому стовпчику таблиці ліній описували би розташування області набором пари координат.

Крім двох уже розглянутих моделей високого рівня на сцену виходить третя, називана об'єктно -орієнтованою моделлю даних. Ця модель включає мову просторових запитів [R.G.Healey, 1991] і відбиває визнання того факту, що потрібен об'єктно-орієнтований доступ і до БД і до виконуємих з нею операціям. Ідеї, що лежать в основі цих систем практично ідентичні об'ектно-орієнтованому підходові в програмуванні [P.Aronsoп, 1987].

2.4 Введення, збереження та редагування БД

з растрових системах головними даними є значення атрибутів осередків растра, що зберігаються в комп'ютері звичайно на твердому диску. Положення кожного осередку растра визначається щодо положень інших осередків растра. З цієї причини редагування зв'язане головним чином із правильним відносним положенням кожного осередку растра.

Якщо растрова система забезпечує зв'язок із зовнішньої СУБД, питання стає трохи складніше в тім, що кожному осередкові растра приєднано кілька різних кодів атрибутів.

з випадку векторів графіка й атрибути зберігаються або як окремі таблиці всередині однієї БД, або як самостійні набори даних, зв'язані набором покажчиків. Поділ графіки й атрибутів вимагає уваги до процедур редагування, застосовуваним до графіки, атрибутам і базам даних. Багато векторні БД дозволяють зберігати окремо частини БД, як великі секції для цілей архівування Ця процедура, називана мозаїчним розміщенням, найчастіше використовується для зменшення обсягу даних, необхідних для одноразового аналізу в дуже великих БД. Хоча мозаїка не вимагає застосування такої формальної схеми, багато хто вважають її корисною для спрощення управління даними.

Найчастіше БД цілком редагується і вичищається перед мозаїчною розбивкою, архівацією і визначенням доступу для відновлення й аналізу. Але так буває не завжди, і тоді вам доведеться вибирати підходящі блоки для редагування. У деяких випадках може знадобитися виконання операції ув'язування по границях блоків для забезпечення стикування частин об'єктів, що перетинають границі блоків.

Загалом, сучасне програмне забезпечення БД, будь те растрове, векторне або на квадро - дереві, забезпечує механізм візуального відображення, що підвищує можливості візуалізації помилок. Конкретні методи будуть залежати від використовуваної моделі даних і складності системи. Оскільки більшість систем дають можливість інтерактивного редагування усередині підсистеми візуалізації, то звичайно мається також і можливість коректувати помилки безпосередньо при виявленні кожної з них.

Хоча деякі помилки можуть відбуватися в результаті недоліків обчислювальних алгоритмів, помилок кодування програм і помилок округлення, і це дійсно трапляється час від часу, усе-таки більшість помилок у БД обумовлені неправильним введенням.

Перший тип помилок відноситься головним чином до векторних систем і називається графічною помилкою. Такі помилки зустрічаються трьох видів: пропуск об'єкта, невірне положення об'єкта і невірний порядок об'єктів.

Другий тип помилок це помилки атрибутів. Вони зустрічаються й у векторних і в растрових системах, з однаковою частотою. Найчастіше вони помилками, а величезний обсяг роботи, що вимагається для великих БД, часто виявляється головним джерелом помилок. У векторних системах помилки атрибутів включають використання невірного коду для атрибута, помилки запису однакових за вимовою, але різних за написанням слів, що унеможливлює вибірку атрибута, якщо в запиті використаний коректний запис. У випадку растра введення найчастіше складається з атрибутів, тому результатом набору невірного коду або приміщення його в невірний осередок растра є карта, що показує ці невірно кодовані осередки в невірних місцях. Такі невірні розташовані атрибутивні дані утворять третій тип помилок, помилки узгодження графіки й атрибутів, що трапляються й у векторних системах, коли вірно набрані коди атрибутів зв'язуються з невірними графічними об'єктами.

Методи, за допомогою яких це буде зроблено, залежать до деякої міри від наявного устаткування і від конкретної системи, що під рукою. Незалежно від того, що за система і як вводять у неї просторові дані, підсистема введення буде мати загальні з іншими характеристики. По-перше, вона спроектована для переносу графічних і атрибутивних даних у комп'ютер. По-друге, вона повинна відповідати хоча б одному з двох фундаментальних методів представлення графічних об'єктів - растровому або векторному. По-третє, вона повинна мати зв'язок із системою збереження і редагування, щоб гарантувати збереження і можливість вибірки того, що введено, і що можна буде усувати помилки і вносити зміни в міру необхідності.

Правило номер один говорить: насамперед визначите, для чого ви створюєте БД . Це щонайменше обмежить уведення даних темами , що швидше за все будуть використовуватися. Тематичні покриття, що вводяться, повинні бути прямо зв'язані з моделюванням і аналізом, що ви маєте намір виконувати, і результатами, що маєте намір одержати.

Необхідність визначення того, які покриття знадобляться в майбутньому, являє собою деяку проблему. Єдиним випадком, коли база даних може створюватися без чіткого розуміння передбачуваного результату (іноді називаного просторово-інформаційним продуктом, є проекти, головна мета яких -

визначити можливі взаємозв'язки між даними тематичних покрить для формулювання початкової наукової гіпотези. Цей підхід не прийнятний для комерційних проектів.

Тому правило номер два, зв'язане з першим, вимагає як можна більш точного визначення цілей перед вибором тематичних покрить.

Правило третє: уникайте використання даних з екзотичних джерел, коли маються звичайні, особливо якщо останні забезпечують подібний рівень точності.

Правило четверте: використовуйте найкращі, найбільш точні дані, необхідні для вашої задачі.

Правило п'яте: вибирайте адекватний рівень точності даних. Більшість тематичних карт містять також інформацію про дороги й інші антропогенні об'єкти, що можуть бути дуже корисними для введення в БД. Скрізь, де можливо, і де їхня якість прийнятна, належить уводити ці дані як окремі покриття з того ж листа карти.

Правило шосте, говорить, що кожне покриття повинне бути як можна більш спеціалізованим. Тобто покриття повинні бути як можна більш спеціалізовані за темами. Чим більш спеціалізоване покриття, тим легше виконувати пошук , якщо треба довідатися щось, що відноситься до даних, які утримуються в одному покритті.

Питання для контролю

1. Типи моделей даних.
2. Основні характеристики і можливості застосування ієрархічної, мережевої й реляційної моделей даних.
3. Особливості багатошарових моделей БД.
4. Введення, збереження та редагування БД БД.
5. Структури баз даних для управління даними.

Лекція 3

Сучасні підходи до створення баз даних

3.1 Реляційні бази даних

переважній більшості СУБД для персональних комп'ютерів інформація організовується у вигляді двомірних таблиць, і їх часто, хоча й не завжди коректно, називають реляційними базами даних.

Реляційні бази даних з'явилися із прагнення перебороти ці недоліки В забезпечити прості й гнучкі способи організації даних. Реляційну базу даних можна визначити як сукупність зв'язаних таблиць. У такій базі інформація організується у вигляді двовимірних таблиць, зі стовпчиками - полями й рядками - записами, у яких зберігаються як самі дані так й інформація про зв'язки між ними.

Звичайно полів і записів незмірно більше. Кожне поле містить найменший елемент даних, якому можна зберігати в таблиці. Це може бути число, слово, якийсь текст або навіть зображення, музика, відеокліп і т.п. Запис (рядок таблиці) являє собою сукупність полів і містить одну копію кожного певного в базі даного поля, навіть у тому випадку, якщо поле не містить ніякої інформації. Безліч записів утворять таблицю. Безліч зв'язаних між собою таблиць утворять **реляційну базу даних**.

Майже всі продукти баз даних, створені з кінця 70-х років, засновані на підході, що називають реляційним; більше того, переважна більшість наукових досліджень в області баз даних протягом останніх 25 років проводилася (можливо, побічно) у цьому напрямку. Реляційний підхід являє собою основну тенденцію сьогоднішнього ринку, і реляційна модель - єдина найбільш істотна розробка в історії розвитку баз даних.

Отже, коротенько, реляційна система - це система, заснована на наступних принципах:

дані для користувача передаються у вигляді таблиць (і ніяк інакше);

користувачеві надаються оператори (наприклад, для вибірки даних), що генерують нові таблиці зі старих. Наприклад, у системі буде оператор одержання

підмножини рядків у даній таблиці й оператор одержання підмножини стовпців - а підмножина рядків і підмножина стовпців, звичайно, можна розглядати як нові таблиці.

• Реляційна база даних — це база даних, сприймана користувачем як набір нормалізованих відносин (тобто змінних відносин) різного ступеня.

Вираження "сприймана користувачем" є вирішальним: ідея реляційної моделі застосовується до зовнішнього й концептуального рівнів системи, а не до внутрішнього рівня. Можна сказати й інакше реляційна модель представляє систему баз даних на рівні абстракції, трохи вилученому від подробиць лежачої в основі цієї системи машини; так само як, наприклад, мова Pascal представляє систему програмування на рівні абстракції, трохи вилученому від подробиць лежачої в основі цієї системи машини. У дійсності реляційну модель можна розглядати як мову програмування, спеціально орієнтовану на додатки баз даних.

Розходження між доменами й (іменованими) відносинами також можна трохи уточнити. Ми називаємо іменовані відносини змінними, тому що їхні значення змінюються згодом. Домени не є змінними в цьому змісті.

традиційних термінах відношення відповідає файлу (логічному, а не фізичному), кортеж — запису (екземпляру, а не типу), атрибут полю (типу,
з не екземпляру). Однак ця відповідність є приблизною. Відношення варто розглядати не як "просто файл", а як файл, що підкоряється певним правилам, це відбивається в значному спрощенні структури об'єктів даних, з якої зіштовхується користувач, і, як наслідок, у спрощенні операторів, необхідних для роботи із цими об'єктами. Якщо говорити точніше, всі дані в реляційній системі представляються одним і тільки одним способом, а саме явними значеннями (цю властивість іноді називають "основним принципом реляційної моделі", а також "інформаційним принципом"). Зокрема, цими явними значеннями представляються логічні зв'язки у відношенні й між відносинами; не існує видимих для користувача покажчиків на файли або записи, не існує видимого для користувача порядку записів, не існує видимих для користувача груп повторення й т.д.

Файли .DBF стандарту dBASE являють собою відображення двовимірної таблиці зі стовпцями - полями й рядками - записами. При пошуку інформації в цих файлах часто доводиться використовувати відомості про положення даних у файлі (номер рядка таблиці, номер запису файлу .DBF) і щодо цього стандарт dBASE не задовольняє вимогам, пропонованим до реляційних баз даних. Поки бази даних на персональних комп'ютерах були відносно невеликі і їх можна було розмістити в одному файлі .DBF, ця обставина не грала істотної ролі, а звична простота таблиці залучала до цього способу організації інформації численних користувачів. Але при збільшенні розмірів баз даних зберігати їх одній таблиці стає неможливим і виникає необхідність виконання інших вимог реляційної моделі.

Відносини типу **один-до-багатьох** використовуються, коли одного запису першої таблиці необхідно поставити у відповідність кілька записів другої таблиці.

Слід зазначити, що визначати відносини між таблицями "у ручну", заняття досить трудомістке. На щастя, цей процес легко може бути автоматизований, або за допомогою макросів Microsoft Access, або за допомогою програми на Visual BASIC.

В дев'яності роки реляційна модель даних перетворилася в основний засіб організації інформації в базах даних не тільки на персональних комп'ютерах, але й на більших ЕОМ. У рамках цієї моделі була розроблена мова структурованих запитів SQL, який став основним засобом для одержання інформації з реляційних баз даних.

3.2 Етапи проектування бази даних

- Визначте **мету** створення бази даних, основні її функції й інформацію, що вона повинна містити. База даних повинна відповідати вимогам тих, хто буде безпосередньо з нею працювати. Для цього потрібно визначити теми, які повинна покривати база даних, звіти, які вона повинна видавати, проаналізувати форми, які в даний момент використовуються для запису даних, порівняти створювану базу даних з добре спроектованою, подібною їй базою.

• Розробіть на папері **структуру таблиць**, які повинна містити база даних. При проектуванні таблиць, рекомендується керуватися наступними основними принципами:

Інформація в таблиці не повинна дублюватися. Не повинне бути повторень і між таблицями. Коли певна інформація зберігається тільки в одній таблиці, то й змінювати її прийдеться тільки в одному місці. Це робить роботу більш ефективною, а також виключає можливість розбіжності інформації в різних таблицях. Кожна таблиця повинна містити інформацію тільки на одну тему. Відомості на кожну тему обробляються набагато легше, якщо утримуються вони в незалежних друг від друга таблицях.

• Визначте **необхідні в таблиці поля**. Кожна таблиця містить інформацію на окрему тему, а кожне поле в таблиці містить окремі відомості по темі таблиці. При розробці полів для кожної таблиці необхідно пам'ятати:

кожне поле повинне бути пов'язане з темою таблиці.

не рекомендується включати в таблицю дані, які є результатом вираження.

у таблиці повинна бути присутня вся необхідна інформація.

• Задайте **ключове поле**. Для того, щоб Microsoft Access міг зв'язати дані з різних таблиць, кожна таблиця повинна містити поле або набір полів, які будуть задавати індивідуальне значення кожного запису в таблиці. Таке поле або набір полів називають основним ключем.

• Визначте **зв'язки між таблицями**. Після розподілу даних по таблицях і визначення ключових полів необхідно вибрати схему для зв'язку даних у різних таблицях. Для цього потрібно визначити зв'язки між таблицями.

• Ще раз перегляните **структуру бази даних** і виявите можливі недоліки. Бажано це зробити на даному етапі, поки таблиці не заповнені даними.

• **Додайте дані й створіть інші об'єкти бази даних**. Якщо структури таблиць відповідають поставленим вимогам, то можна вводити всі дані. Потім можна створювати будь-які запити, форми, звіти, макроси й модулі.

у Використовуйте **засоби аналізу** в Microsoft Access. В Microsoft Access існує два інструменти для вдосконалення структури баз даних. Майстер аналізу таблиць досліджує таблицю, якщо буде потреба пропонує нову її структуру й зв'язки, а також переробляє її. Аналізатор швидкодії досліджує всю базу даних, дає рекомендації з її поліпшення, а також здійснює їх.

3.3 Проектування БД: загальні положення

Крім основної ідеї просторово-інформаційних продуктів існують і інші сторони проектування БД для об'єкта. Коли маються кілька проектів, необхідно розглядати відповідні досліджувані області по окремості. Вони найчастіше ґрунтуються на формальному рішенні про те, чому область піддається вивченню. Так, для дослідження міського рівня, усе місто буде областю дослідження. Для дослідження, зв'язаного з природними явищами або об'єктами, як, наприклад, забруднення ріки, весь водозбірний басейн ріки може бути обраний як область вивчення, оскільки забруднення може приходити з будь-якої території, що дає стік в один із припливів ріки. Коротше кажучи, область вивчення може вибиратися на основі політичних і адміністративних границь, фізичних границь, границь власності, або відбивати, головним чином, фінансові обмеження або обмеження за даними, що виявляються на ранніх стадіях процесу проектування.

Якщо для деякої частини передбачуваної області вивчення маються більш докладні дані, чим для іншої частини, вони не повинні диктувати розмір області вивчення. Цю обставину можна використовувати таким чином, що невелика частина з більш докладними даними може відігравати роль прототипу для детального аналізу всієї області вивчення, у розрахунку на те, що надалі і на всю область можна буде одержати дані такої ж подробиці. Крім того, вони можуть придатися для поліпшення знання про всю область вивчення.

Коли планується використовувати який-небудь вид інтерполяції, границя області вивчення повинна бути розширена в достатньому ступені, щоб забезпечити коректні результати інтерполяції на її краях . І, нарешті, чим більше область вивчення, тим більше грошей і часу буде потрібно на створення бази даних. Часто саме питання вартості є головним обмежником у встановленні

границь області вивчення. Крім цього, розширення області вивчення підвищує імовірність недоліку даних для всіх покрить, необхідних для проведення аналізу.

Масштаб, дозвіл і рівень детальності. Існує зв'язок між розміром досліджуваної області і масштабом карт, що вводяться. Чим дрібніше масштаб карти, тим вище ступінь генералізації, тим менш точне представлення об'єктів реального світу. На вибір масштабу впливає важливість відповідного покриття для моделей аналізу. Хоча не існує загальних указівок для визначення підходящого масштабу, більшість професіоналів використовують підхід "найкращих доступних даних", вважаючи, що краще більше подробиць, чим менше.

Растрові дані мають особливість: зі зменшенням розміру осередків растра швидко зростає обсяг даних. І хоча сучасні комп'ютери мають досить ємні пристрої збереження, великий обсяг даних значно сповільнює виконання операцій над покриттями, що використовують пошук по околиці.

і одних випадках розмір осередків растра може диктуватися найменшим об'єктом, що представляється, в інші - вимогами моделі [DeMers, 1992], у третіх - питаннями сумісності з іншими цифровими даними.

Класифікація. При проектуванні БД треба розглядати не тільки наявні джерела даних, але і систему класифікації, що задовольняє вимогам моделювання. Рішення про останнє краще приймати після розгляду видів даних, що вводяться. Використання більш докладної класифікації часто переважно по двум причинам; вона дає користувачеві більший обсяг зведень, і при порівнянні з даними іншого покриття меншої подробиці завжди можна агрегувати деякі класи, у той час як зворотний процес або утруднений, або зовсім неможливий.

Але класифікація - більше чим просто вибір належного рівня детальності. Потрібно врахувати, що робиться порівняння між покриттями, і, таким чином, між класифікаціями. Крім того, часто необхідно мати можливість проводити погоджену класифікацію в межах одного покриття даних з різних джерел. Допустимо, потрібно покриття з класами ґрунтів регіонального охоплення, створене на основі різних досліджень ґрунтів більш детального окружного рівня. Якщо ці дослідження проводилися на значному тимчасовому видаленні друг від

друга, то буде необхідно переробити них, щоб домогтися однорідності класифікації Бэрроу [P.A.Burrough, 1986].

Система координат і проекція. Вибір проекції визначається площею області вивчення і доступних даних. При цьому варто враховувати, що перетворення проекцій вносять додаткову погрішність у дані. На вибір проекції впливає також те, які характеристики земної поверхні повинні зберігатися (звичайно ті, що найбільш важливо для аналізу). Загалом, і стосовно системи координат, і стосовно проекції, просторова і тимчасова сумісність дуже важливі для коректності процесу прийняття рішень.

Введення даних в БД. У випадку БД одна тільки побудова бази даних часто займає три чверті часу. У комерційних додатках це означає, що три чверті вартості системи також піде на цю операцію і редагування. Є багато способів уведення даних. Одні виглядають примітивними, начебто приміщення прозорої сітки на карту. Інші - більш сучасні, тому що використовують пристрою цифрового введення — дигітайзери і сканери.

Питання для контролю

1. Підходи до проектування реляційних баз даних.
2. Методи і моделі, за допомогою яких виконується опис и структуруються різноманітні предметові середовища.
3. Класична методологія проектування баз даних.
4. Етапи проектування баз даних.
5. Реляційні бази даних, їх характеристика

.

Лекція 4

Концептуальна модель бази даних

4.1 Концептуальна модель організації даних

Концептуальна модель даних слугує винятково для проектування і виступає як засіб відображення уявлень людини про системи та процеси предметної області реального світу , сприяє їхньому розумінню. Вона описує об'єкти предметної області та їхні взаємозв'язки на основі сформованих загальносистемних принципів і вимог до організації інформаційних баз без зазначення засобів їхнього фізичного зберігання [21].

Проектування інформаційних баз передбачає три основних рівні організації даних — концептуальний, логічний і фізичний.

Першим етапом розробки будь-якої інформаційної системи має стати побудова **концептуальної моделі організації даних,** як засобу точного відображення уявлень людини про системи та процеси предметної області реального світу. При формуванні концептуальної моделі основна увага спрямовується на структуризацію даних і виявлення взаємозв'язків між ними без розгляду особливостей реалізації та ефективності обробки.

Предметною областю дорадчих систем, побудованих на базі БД, є територія з усією притаманною їй специфікою природних умов, ресурсним потенціалом, поширеними в її межах видами господарської діяльності.

Для організації даних важливі не тільки взаємозв'язки, що здійснюються у просторових системах реального світу, але й заснована на них інформаційна взаємодія між структурно-функціональними одиницями різних рангів у самій інформаційній базі. Тому використання БД як засобу вирішення завдань у територіальному плані потребує, з одного боку, формалізації інформації, з іншого , її структуризації на основі визначення об'єктів інформаційної діяльності.

4.2 Структура і технологія наповнення

Структуру і склад інформаційної бази на різних рангах інтеграції даних з визначенням методів фіксації показників та форм представлення інформації обгрунтовано на основі сформованих систем вимог і концептуальних моделей організації даних у різних функціональних модулях політики конфіденційності (ПІК) об'єкту, які дають можливість створення адекватної інформаційної моделі території досліджень.

При цьому системну організацію тематичної інформації здійснюють, як було зазначено вище, з урахуванням адресних вимог бази знань і бази даних.

База знань формується як сукупність знань для обгрунтування та оптимізації прийняття рішень. Концептуальна модель організації даних у її межах передбачає диференціацію інформації з видокремленням нормативно-регламентуючої бази (НРБ) і системи загальнодовідкових даних, що забезпечує вибір параметрів на стадії ідентифікації об'єктів.

Нормативно-регламентуюча база. Одним із перших етапів створення бази знань є розробка нормативно-регламентуючої бази з системою кодифікації та відповідними експертними системами, що забезпечують вибір оптимальних умов вирощування тієї чи іншої сільгоспкультури, визначення складу і параметрів технологічних операцій рослинництва.

В її основу покладено структуризацію даних у середині кожного з тематичних та системних модулів ПС-1, ПС-2, ПС-3 з використанням класифікаційних схем як засобів описування вміщеної у них інформації.

Основою формування інформаційної бази при вирішенні завдань просторового оцінювання має бути територіально-галузева інформаційна база та природно-ресурсний паспорт об'єкта, який вміщує елементи інформаційно-довідкової і інформаційної баз даних[8].

Створення регіональних баз знань, локальних інформаційно-довідкових і відповідно інформаційних баз даних здійснюється в автономному режимі для різних рівнів функціональних завдань на основі єдиних концептуальних моделей організації і структуризації інформації в рамках ПІК.[8]

Концептуальне подання — це подання всієї інформації бази даних у трохи більш абстрактній формі у порівнянні з фізичним способом зберігання даних. Однак концептуальне подання істотно відрізняється від способу подання даних якому-небудь окремому користувачеві. Загалом кажучи, концептуальне подання - це подання даних такими, які "вони є насправді", а не такими, якими змушений їх бачити користувач у рамках, наприклад, певного мови або використовуваного апаратного забезпечення. Концептуальне подання складається з безлічі екземплярів кожного типу концептуального запису. Концептуальний запис зовсім не обов'язково повинна збігатися із зовнішнім записом, з одного боку, і зі збереженим записом - з іншої.

Концептуальне подання визначається за допомогою концептуальної схеми, що включає визначення кожного типу концептуальних записів. Концептуальна схема використовує іншу мову визначення даних — концептуальну. Щоб домогтися незалежності даних, не можна включати у визначення концептуальної мови будь-який розгляд структури зберігання або методу доступу. Визначення концептуальної мови повинні ставитися тільки до змісту інформації. Це означає, що в концептуальній схемі не повинно бути ніякого згадування про подання збереженого файлу, послідовності збережених записів , індексуванні , хеш-адресації, покажчиках або інших подробицях зберігання або доступу. Якщо концептуальна схема дійсно забезпечує незалежність даних у цьому змісті, то зовнішні схеми, певні на основі концептуальної, свідомо будуть забезпечувати незалежність даних.

Концептуальне подання — це подання всього вмісту бази даних, а концептуальна схема — це визначення такого подання. Однак сьогоднішня система реально не підтримує такого концептуального рівня, що хоча б небагато наблизився до цього ступеня розвиненості; у більшості існуючих систем "концептуальна схема" у дійсності являє собою трохи більше, ніж просте об'єднання всіх окремих зовнішніх схем з додатковими засобами безпеки й правилами забезпечення цілісності [10].

4.3 Відображення

Відображення концептуальний-внутрішній визначає відповідність між концептуальним поданням і збереженою базою даних, тобто як концептуальні записи й поля представлені на внутрішньому рівні. При зміні структури збереженої бази даних, тобто при внесенні змін у визначення структури зберігання, змінюється й відображення концептуальний-внутрішній таким чином, щоб концептуальна схема залишилася незмінною. Інакше кажучи, щоб зберегти незалежність даних, результати таких змін не повинні торкнутися концептуального рівня.

Відображення зовнішній-концептуальний визначає відповідність між деяким зовнішнім поданням і концептуальним поданням. Загалом, розходження, які можуть існувати між цими двома рівнями, подібні до розходжень між концептуальним поданням і збереженою базою даних.

Між іншим, більшість систем дозволяє виражати визначення одного зовнішнього подання через інше (тобто за допомогою відображення зовнішній-зовнішній) не вимагаючи обов'язково явно визначати відображення на концептуальний рівень. Ця можливість корисна, якщо кілька подань схожі між собою. Зокрема, ця можливість є в багатьох реляційних системах.

Первинний ключ - це унікальний ідентифікатор для таблиці, тобто стовпець або така комбінація стовпців, що в будь-який момент часу не існує двох рядків, що містять однакове значення в цьому стовпці або комбінації стовпців.

з нарешті, **домен** - це загальна сукупність значень, з якої беруться справжні значення для певних атрибутів певного відношення.

Домени насамперед мають концептуальну природу. Вони можуть бути або не бути явно збережені в базі даних як реальні набори значень; фактично в більшості випадків вони не зберігаються. Але вони повинні бути, принаймні, визначені в рамках визначень бази даних і тоді кожне визначення атрибута повинне включати посилання на відповідний домен, у такий спосіб системі буде відомо, які атрибути можна порівнювати, а які – немає.

Щоб конкретизувати ідеї реляційної моделі використовується гіпотетично реляційна мова для ілюстрації цих ідей. Ясно, що в першу чергу потрібний спосіб

створення нового домена: CREATE DOMAIN domain data-type ; тут domain— це ім'я нового домена, а data-type відповідає типу даних [7].

4.4 Інфологічна модель даних " Сутність-Зв'язок"

4.4.1 Основні поняття

Мета інфологічного моделювання - забезпечення найбільш природних для людини способів збору й подання тої інформації, що передбачається зберігати в створюваній базі даних . Тому інфологічна модель даних намагаються будувати за аналогією із природною мовою (остання не може бути використаний у чистому виді через складність комп'ютерної обробки текстів і неоднозначності будь-якої природної мови). Основними конструктивними елементами інфологічних моделей є сутності, зв'язки між ними і їхньої властивості (атрибути).

Сутність – будь-який помітний об'єкт (об'єкт , що ми можемо відрізнити від іншого), інформацію про яке необхідно зберігати в базі даних. Сутностями можуть бути люди, місця, літаки, рейси, смак, колір і т.д. Необхідно розрізняти такі поняття, як тип сутності й екземпляр сутності. Поняття тип сутності ставиться до набору однорідних особистостей, предметів , подій або ідей, що виступають як ціле. Екземпляр сутності ставиться до конкретної речі в наборі. Наприклад, типом сутності може бути МІСТО, а екземпляром - Херсон, Київ і т.д.

Атрибут – пойменована характеристика сутності. Його найменування повинне бути унікальним для конкретного типу сутності, але може бути однаковим для різного типу сутностей. Атрибути використовуються для визначення того, яка інформація повинна бути зібрана про сутність. Абсолютне розходження між типами сутностей і атрибутами відсутній. Атрибут є таким тільки у зв'язку з типом сутності. В іншому контексті атрибут може виступати як самостійна сутність.

Ключ – мінімальний набір атрибутів, за значеннями яких можна однозначно знайти необхідний екземпляр сутності. Мінімальність означає, що виключення з набору будь-якого атрибута не дозволяє ідентифікувати сутність по тому що залишилися.

Зв'язок – асоціювання двох або більше сутностей. Якби призначенням бази даних було тільки зберігання окремих, не зв'язаних між собою даних, то її структура могла б бути дуже простою. Однак одне з основних вимог до організації бази даних - це забезпечення можливості відшукання одних сутностей за значеннями інших, для чого необхідно встановити між ними певні зв'язки. А тому що в реальних базах даних нерідко втримуються сотні або навіть тисяча сутностей, те теоретично між ними може бути встановлене більше мільйона зв'язків. Наявність такої безлічі зв'язків і визначає складність інфологічних моделей.

Існують три основні класи сутностей: стрижневий, асоціативний й характеристичний, а також підклас асоціативних сутностей – позначення.

Стрижнева сутність (стрижень) – це незалежна сутність (трохи докладніше вона буде визначена нижче).

Асоціативна сутність (асоціація) – це зв'язок виду " багато-до-багатьох" ("-до-багатьох" і т. д.) між двома або більше сутностями або екземплярами сутності . Асоціації розглядаються як повноправні сутності:

6. вони можуть брати участь в інших асоціаціях і позначеннях точно так само, як стрижневі сутності;

7. можуть мати властивості, тобто мати не тільки набір ключових атрибутів, необхідних для вказівки зв'язків, але й будь-яке число інших атрибутів, що характеризують зв'язок.

Характеристична сутність (характеристика) – це зв'язок виду " багато - до-однієї" або " одна-до-однієї" між двома сутностями. Єдина мета характеристики в рамках розглянутої предметної області складається в описі або уточненні деякої іншої сутності. Необхідність у них виникає у зв'язку з тим, що сутності реального миру мають іноді багатозначні властивості.

4.4.2 Первинні й зовнішні ключі

Нагадаємо, що ключ або можливий ключ – це мінімальний набір атрибутів, за значеннями яких можна однозначно знайти необхідний екземпляр сутності. Мінімальність означає, що виключення з набору будь-якого атрибута не дозволяє

ідентифікувати сутність по тому що залишилися. Кожна сутність володіє хоча б одним можливим ключем. Один них приймається за первинний ключ. При виборі первинного ключа варто віддавати перевагу нескладовим ключам або ключам, складеним з мінімального числа атрибутів. Недоцільно також використовувати ключі з довгими текстовими значеннями. Також необхідно забезпечити унікальність первинного ключа.

Зовнішні ключі:

Якщо сутність У зв'язує сутності А та В, то вона повинна включати зовнішні ключі, що відповідають первинним ключам сутностей А та В.

Якщо сутність В позначає сутність А, то вона повинна включати зовнішній ключ, що відповідає первинному ключу сутності А.

будь-якій таблиці, як правило, хоча б одне поле є **ключовим**. Завдяки ключовим полям будь-який запит до бази даних може бути оформлений у вигляді операцій з таблицями і їхніми полями (атрибутами), але не з їхніми рядками. Крім того, ключові поля використовуються для підтримки цілісності реляційної бази даних. Таблиця може мати тільки один унікальний ідентифікатор, називаний **первинним ключем (**primary key**)** і трохи вторинних (зовнішніх) ключів (foreign key), що посилаються на первинні ключі інших таблиць. При цьому зовнішні ключові поля таблиці, повинні мати одне зі значень, певне для первинних ключів.

Назва полів у допоміжній таблиці можуть і не збігатися з назвами первинних ключів, але тип даних і розмір цих полів повинні збігатися обов 'язково. Забезпечення цілісності даних досягається при встановленні двох перемикачів у меню СУБД Access – «Каскадне відновлення зв'язаних полів» і «Каскадне видалення зв'язаних полів»[11].

4.5 Нормалізація відносин

Ті самі дані можуть групуватися в таблиці (відносини) різними способами, тобто можлива організація різних наборів відносин взаємозалежних інформаційних об'єктів. Угруповання атрибутів у відносинах повинне бути

раціональним, тобто мінімізуючим дублювання даних і процедури, що спрощує, їхню обробку й відновлення.

Певний набір відносин має кращі властивості при включенні, модифікації, видаленні даних, чим всі інші можливі набори відносин, якщо він відповідає вимогам нормалізації відносин.

Нормалізація відносин — формальний апарат обмежень на формування відносин (таблиць), що дозволяє усунути дублювання, забезпечує несуперечність збережених у базі даних, зменшує працевитрати на ведення (уведення, коректування) бази даних.

Виділено три **нормальні форми відносин** і запропонований механізм, що дозволяє будь-яке відношення перетворити до третього (самої зробленої) нормальній формі.

Перша нормальна форма.

Відношення називається нормалізованим або наведеним до **першої нормальної форми**, якщо всі його атрибути прості (неподільні). Перетворення відносини до першої нормальної форми може привести до збільшення кількості реквізитів (полів) відносини й зміні ключа.

Друга нормальна форма.

Щоб розглянути питання приведення відносин до другої нормальної форми, необхідно дати пояснення до таких понять, як функціональна залежність і повна функціональна залежність.

Описові реквізити інформаційного об'єкта логічно пов'язані із загальним для них ключем, цей зв'язок носить характер **функціональної залежності** реквізитів.

Функціональна залежність реквізитів — залежність, при якій екземплярі інформаційного об'єкта певному значенню ключового реквізиту відповідає тільки одне значення описового реквізиту.

Таке визначення функціональної залежності дозволяє при аналізі всіх взаємозв'язків реквізитів предметної області виділити самостійні інформаційні об'єкти.

у випадку складеного ключа вводиться поняття **функціонально повної** залежності.

Функціонально повна залежність не ключових атрибутів полягає в тім, що кожний не ключовий атрибут функціонально залежить від ключа, але не перебуває у функціональній залежності ні від якої частини складеного ключа.

Відношення буде перебувати в другій нормальній формі, якщо воно перебуває в першій нормальній формі, і кожний не ключовий атрибут функціонально повно залежить від складеного ключа.

Третя нормальна форма.

Поняття третьої нормальної форми ґрунтується на понятті **нетранзитивної** залежності.

Транзитивна залежність спостерігається в тому випадку, якщо один із двох описових реквізитів залежить від ключа, а інший описовий реквізит залежить від першого описового реквізиту.

Відношення буде перебувати в третій нормальній формі, якщо воно перебуває в другій нормальній формі, і кожний неключовий атрибут нетранзитивно залежить від первинного ключа.

Для усунення транзитивної залежності описових реквізитів необхідно провести "розщеплення" вихідного інформаційного об'єкта. У результаті розщеплення частина реквізитів віддаляється з вихідного інформаційного об'єкта й включається до складу інших (можливо, знову створених) інформаційних об'єктів.

Типи зв'язків:

Всі інформаційні об'єкти предметної області зв'язані між собою. Розрізняються **зв'язки** декількох типів, для яких уведені наступні позначення:

один до одного (1:1);

один до багатьох (1 :
M); багато до багатьох (M :
M).

Зв'язок **один до одного** (1:1) допускає, що в кожний момент часу одному екземпляру інформаційного об'єкта А відповідає не більше одного екземпляра інформаційного об'єкта В й навпаки.

При зв 'язку **один до багатьох** (1:M) одного екземпляра інформаційного об'єкта А відповідає 0, 1 або більше екземплярів об'єкта В, але кожний екземпляр об'єкта У зв'язаний не більш ніж з 1 екземпляром об'єкта А.

Зв'язок **багато до багатьох** (M:M) допускає, що в кожний момент часу одному екземпляру інформаційного об'єкта А відповідає 0, 1 або більше екземплярів об'єкта В й навпаки [25].

Питання для контролю

1. Два підходи до моделювання предметної області.
2. Технологія аналізу предметної області.
3. Поняття сутностей, їх атрибутів, зв’язків між ними.
4. Поняття первинних та зовнішніх ключів.
5. Поняття нормальні форми.

.

Лекція 5

Логічна та фізична моделі баз даних

5.1 Основні поняття

Логічна модель даних відображує концептуальну модель у вигляді структур, що використовуються системою управління базами, тобто це формалізоване подання концептуальної моделі. **Реляційна модель** — один із видів логічної моделі, в якій об'єкти взаємозв'язку представляють за допомогою таблиць. При цьому взаємозв'язки також розглядаються як об'єкти. Кожна таблиця відображує один об'єкт і складається із рядків та стовпців [21].

Фізична модель даних визначає порядок розміщення даних на зовнішніх носіях ЕОМ, формат їхнього зберігання та конкретні способи доступу до них; це цифрове подання даних, що зберігаються у фізичних пристроях комп'ютерів [21].

Моделі даних ґрунтуються на двох основних видах диференціації або групування інформації: тематичному і системному.

Тематична організація інформації (тематичні дані) — групування даних за характеристиками основних компонентів систем або предметної області, що здійснюються за принципом змістовної організації інформації

1 Предметна область — це частина реального світу, в межах якої визначається набір даних і методів маніпулювання ними для вирішення конкретних завдань або виконання досліджень.

Експертні системи забезпечують регламентацію правил виконання того чи іншого завдання, у тому числі з оптимізації та нормування, їхньої формалізації і потребують розробки комплексу програм, що дають змогу приймати рішення за допомогою комп'ютера. Для їхнього функціонування необхідна наявність трьох основних компонентів — фактів, правил та структур управління.

Фактичні знання повідомляються експертній системі експертом у процесі діалогу і відображують погляди людини щодо інтерпретації даних на момент роботи. Процедурні знання тісно пов'язані з фактичним накопиченим досвідом, на основі якого відпрацьовувались правила, що регламентують поведінку системи. Знання, на основі яких здійснюють управління, дають змогу підбирати найкращу

стратегію у роботі системи. Аналітична частина ЕС у вигляді системи SQL - запитань являє собою алгоритми так званих продукцій, побудованих на виразах «якщо — то», логічних описів у відповідних модулях[8].

Системи управління базами даних призначені для зберігання й управління всіма типами даних. СУБД оптимізовані для подібних завдань, тому в багатьох БД вбудована підтримка СУБД. Ці системи не мають подібних з БД інструментів для аналізу й візуалізації.

До апаратного забезпечення системи відносяться:

накопичувачі для зберігання інформації (звичайно диски з переміщуваними головками) разом із приєднаними пристроями уведення-виводу, контролерами пристроїв, каналами уведення-виводу й т. п.;

процесор (або процесори) разом з основною пам'яттю, що використовується для підтримки роботи програмного забезпечення системи.

Між властиво **фізичною** базою даних (тобто даними, які в дійсності збережені) і користувачами системи розташовується рівень програмного забезпечення— диспетчер бази даних (database manager) або, що більш звично, система управління базами даних, СУБД (database management system (DBMS)). Всі запити користувачів на доступ до бази даних обробляються СУБД; можливості додавання файлів (або таблиць), вибірки відновлення даних у цих файлах або таблицях також забезпечує СУБД. Основна функція, виконувана СУБД, — це надання користувачеві бази даних можливості працювати з нею, не вникаючи в деталі на рівні апаратного забезпечення (користувач більше відсторонений від цих деталей, чим прикладний програміст, що використовує середовище програмування). Іншими словами, СУБД дозволяє користувачеві розглядати базу даних як об'єкт більше високого рівня в порівнянні з апаратним забезпеченням, а також підтримує користувальницькі операції, що виражаються в термінах високого рівня, (наприклад, операції, які можна виконувати за допомогою мови SQL.

СУБД- найбільш важливий, але не єдиний програмний компонент системи. Серед інших - утиліти, засоби розробки додатків, засоби проектування, генератори звітів і ін [10].

5.2. Бази даних і системи управління базами даних

Для маніпулювання інформацією (введенням, пошуком і т.п.) використовуються спеціальні пакети програм, що мають назву **системи управління базами даних** (СУБД). Цей вид програмного забезпечення в останні роки дуже швидко вдосконалюється. З однієї сторони СУБД усе ширше використовуються для маніпулювання новими типами інформації (мультімедіа , БД- системи і т.п.) З іншого боку - створені нові технології (архітектура " клієнт-сервер", розподілені бази даних, гіпертекст і т.п.), які дозволяють забезпечити доступ до інформації широкому колу користувачів рамках мережі Internet, відкриваючи тим самим принципово нові можливості для вивчення навколишнього середовища.

останні роки, завдяки розвитку технологій мультимедіа, за допомогою комп'ютерів стало можливим обробляти практично будь-які типи інформації про навколишнє середовище - замальовки, звуки, відео, і термін «інформація» став часто використовуватися як синонім терміна «дані».

Системи управління базами даних (СУБД) з'явилися раніше, ніж персональні комп'ютери. Специфіка великих ЕОМ, на яких відбувалося їхнє становлення в сімдесяті роки, багато в чому визначило особливості тих СУБД - вони дозволяли кваліфікованому програмістові робити дуже багато, починаючи від обробки транзакцій і закінчуючи перенесенням даних і програм в інші операційні системи й на інші ЕОМ (платформи). Але ці СУБД (Oracle, INGRES і т.п.) не стали стандартом для персональних комп'ютерів, тому що пред'являли занадто високі вимоги до характеристик використовуваної обчислювальної техніки й до кваліфікації користувачів.

вісімдесяті роки було розроблено велике число СУБД спеціально для персональних комп'ютерів. У нашій країні найбільше поширення одержали такі СУБД, як FoxBASE, dBASE III plus, R:Base, Paradox, а наприкінці вісімдесятих років придбав популярність пакет Clipper. Слід зазначити, що FoxBASE, dBASE III plus і Clipper використовували ті самі принципи організації інформації й були

сумісні на рівні файлів баз даних, тому іноді всі ці системи розглядали як модифікації dBASE III plus.

Система програмування **dBASE III plus** була розроблена фірмою Ashton-Tate на основі своїх більш ранніх СУБД - dBASE II і dBASE III. В dBASE III plus основна увага була приділена вдосконалюванню користувальницького інтерфейсу (режим ASSIST), що істотно спростило процедуру створення й модифікації баз даних, сортування й індексацію записів. Створення й використання досить складних структур баз даних було можливо безпосередньо з режиму ASSIST без складання прикладних програм мовою dBASE, що робило цю СУБД доступною для широкого кола користувачів. Це забезпечило величезну популярність dBASE III plus, і наприкінці вісімдесятих років ця СУБД була фактичним стандартом для реляційних баз даних, незважаючи на деякі недоліки, властивій цій системі.

Однак до початку 90-х років ситуація змінилася. Найбільш популярними СУБД для ПК сталі FoxBASE (FoxPro) і Paradox. СУБД FoxPro була розроблена фірмою

Fox Software Inc. у першу чергу для створення додатків для користувачів. Ця СУБД мала дуже потужні програмні засоби й дозволяла легко писати прикладні програми, хоча, бути може, і не мала такого дружнього інтерфейсу, як dBASE III plus. СУБД Paradox була розроблена фірмою Borland International і також була більше орієнтована на створення додатків на основі убудованої повнофункціональної мови програмування PAL. У цієї СУБД використовувався новий метод організації інформації, заснований на метафорі таблиці, що дозволяло з однієї сторони легко реалізувати систему запитів за зразком (Query by Example), а з іншого боку забезпечити дуже високу швидкодію при пошуку інформації.

Одним з недоліків СУБД FoxPro, dBASE III plus, Paradox була неможливість створення з їхньою допомогою файлів .EXE, що автономно

працюють під управлінням DOS. Саме тому широке поширення (у всякому разі в нашій країні) придбав пакет Clipper фірми Nantucket, що із самого початку призначався для компіляції прикладних програм. Clipper працював файлами .DBF, забезпечуючи досить високу швидкодію. У теж час це була відкрита система, що

дозволяла розширювати можливості мови за рахунок додатків, написаних на інших мовах програмування - Assembler'е й С.

1991 р., коли фірма Ashton-Tate була придбана фірмою Borland International, фірма Fox Software Inc. - MicroSoft, а фірма Nantucket -

Computer Associates, формально почався новий етап у розвитку СУБД для ПК, хоча основні ідеї цього етапу пророблялися значно раніше. Можна виділити три основні особливості нового етапу: розподілені бази даних, графічний користувальницький інтерфейс і архітектура " клієнт-сервер". В 1993 р. перші СУБД цього нового покоління - Microsoft Access, dBase IV і т.п. з'явилися на ринку й одержали широке поширення серед користувачів персональних комп'ютерів.

У цей час фактичним стандартом систем управління базами даних для персональних комп'ютерів є СУБД **Microsoft Access**. Пакет Microsoft Access for Windows 95 є потужним засобом управління базами даних, що підтримує реляційну модель даних і дозволяє створювати складні додатки на особливому діалекті Visual BASIC (VBA). Microsoft Access можна застосовувати для пошуку й обробки всіляких даних, а також для підготовки звітних документів. Користувальницький інтерфейс досить простий і надає користувачеві зручні можливості для маніпулювання базами даних, так що освоєння пакета звичайно не викликає складностей.

У зв'язку з бурхливим розвитком мережі Internet, яка є гігантською розполільною базою даних, зріс інтерес до таким СУБД як **Oracle**. У цей час ця система управління базами даних установлена на багатьох серверах Мережі.

Система управління базами даних Oracle фірми Oracle є одним з лідерів ринку багатоплатформенних СУБД. Вона може працювати на більш ніж двохстах типах ЕОМ, включаючи ПК типу IBM PC і Apple Macintosh. У програмне забезпечення цієї СУБД входить одна з найбільш повних реалізацій мови структурованих запитів SQL, а також генератори меню, звітів і інших екранних форм. Крім того, програмне забезпечення дозволяє на підставі інформації, що зберігається в СУБД, будувати більше 50 типів графіків і діаграм. Oracle містить дуже надійну систему захисту даних, їхньої цілісності й несуперечності [29].

5.3 Характеристика ACCESS

Access — це, насамперед, система управлінняння базами даних (СУБД). Як і інші продукти цієї категорії, вона призначена для зберігання й пошуку даних, подання інформації в зручному виді й автоматизації часто повторюваних операцій (таких, облік, планування й т.п.). За допомогою Access можна розробляти прості й зручні форми уведення даних, а також здійснювати обробку даних і видачу складних звітів.

Access - потужний додаток Windows; уперше продуктивність СУБД органічно сполучається з тими зручностями, які є в розпорядженні користувачів Microsoft Windows. Оскільки обоє ці продукти- дітища компанії Microsoft, вони прекрасно взаємодіють між собою . Система Access працює під управлінням Windows 2000 або Windows NT, так що при роботі з нею користувачеві доступні всі переваги Windows. Можна вирізати, копіювати й вставляти дані з будь-якого додатка Windows в Access і навпаки; можна створити проект форми в Access і вставити його в конструктор форм.

За допомогою об'єктів OLE (Object Linking and Embedding - зв'язування у впровадження об'єктів) в Windows 2000 і компонентах Microsoft Office 2000 (Excel, Word, PowerPoint і Outlook) можна перетворити Access у справжнє операційне середовище баз даних. За допомогою нових розширень для Internet можна створювати форми, які будуть прямо взаємодіяти з даними з World Wide Web, і транслювати їх у подання мовою HTML, що забезпечує роботу з переглядачами web сторінок.

При всьому цьому Access — не просто СУБД. Як реляційна СУБД Access забезпечує доступ до всіх типів даних і дозволяє використовувати одночасно кілька таблиць бази даних. При цьому можна істотно спростити структуру даних, полегшуючи тим самим виконання поставлених завдань. Таблицю Access можна зв'язати з даними, що зберігаються на великий ЕОМ або на сервері. З іншого боку, можна використовувати таблиці, створені в середовищі Paradox або dBASE. Отримані результати можна швидко й легко зв'язати й об'єднати з даними з електронних таблиць Excel. Працюючи в середовищі Microsoft Office 2000,

користувач одержує у своє розпорядження повністю сумісні між собою Access і Word, Excel і PowerPoint.

Система Access - це набір інструментів кінцевого користувача для управління базами даних. У її состав входять конструктори таблиць, форм, запитів і звітів. Цю систему можна розглядати і як середовище розробки додатків. Використовуючи макроси або модулі для автоматизації рішення завдань, можна створювати орієнтовані на користувача додатки такіж потужні, як і додатки, написані безпосередньо на мовах програмування. При цьому вони будуть включати кнопки, меню й діалогові вікна. Програмуючи мовою VBA, можна створювати такі потужні програми, як сама система Access.

Потужність і доступність Access роблять цю систему кращою СУБД із представлених сьогодні на ринку.

Справжня реляційна модель баз даних Access повною мірою реалізоване управління реляційними базами даних. Система підтримує первинні й зовнішні ключі й забезпечує цілісність даних на рівні ядра (що запобігає несумісним операціям відновлення або видалення даних). Крім того, таблиці в Access постачені засобами перевірки допустимості даних, що запобігають некоректному введенню поза залежністю від того, як він здійснюється, а кожне поле таблиці має свій формат і стандартні описи, що істотно полегшує введення даних. Access підтримує всі необхідні типи полів, у тому числі текстовий, числовий, лічильник, грошовий, дата/час, MEMO, логічний, гіперпосилання й поля об'єктів OLE. Якщо в процесі спеціальної обробки в полях не виявляється ніяких значень, система забезпечує повну підтримку порожніх значень.

Реляційна обробка даних в Access за рахунок гнучкої архітектури системи здатна задовольнити будь-які потреби. При цьому Access може використовуватися як автономна СУБД у режимі файл-сервера або клієнтського компонента таких продуктів, як SQL Server. Крім того, Access підтримує протокол ODBC (Open Database Connectivity), що дозволяє підключатися до баз дані безлічі різних форматів, таких як SQL Server, Oracle, Sybase і навіть DB/2 для більших ЕОМ фірми IBM.

Система Access підтримує обробку транзакцій з гарантією їхньої цілісності. Крім того, передбачений захист на рівні користувача, що дозволяє контролювати доступ до даних окремих користувачів і цілих груп.

Прості у використанні майстри й конструктори Майстер (Wizard) може перетворити години роботи в лічені мінути. Майстри задають навідні запитання щодо змісту, стилю й формату створюваного об'єкта; потім вони автоматично будують потрібний об'єкт. У складі Access біля ста майстрів, що допомагають конструювати бази даних, додатки, таблиці, форми, звіти, діаграми, поштові наклейки, елементи управління й властивості. Допускається навіть настроювання майстрів для рішення різних завдань.

Імпортування, експортування й зв'язування зовнішніх файлів Access дозволяє імпортувати й експортувати файли багатьох відомих форматів, включаючи dBASE, FoxPro, Excel, SQL Server, Oracle, Btrieve, багато текстових форматів ASCII (у тому числі з фіксованою довжиною рядка або заданим обмежником), а також дані у форматі HTML. У результаті імпортування створюється таблиця Access; у результаті експортування таблиці Access створюється файл у заданому форматі.

Зв'язування (раніше йменувалося приєднанням) означає, що можна використовувати зовнішні дані без створення таблиці Access. Можна встановлювати подібний зв'язок з даними dBASE, FoxPro, Excel, ASCII і SQL. Дуже потужна можливість - зв'язування таблиць Access з їхніми зовнішніми таблицями з наступним спільним використанням; це ставиться до таблиць Access, dBASE, FoxPro і SQL Server.

Форми й звіти WYSIWYG. Вікна конструкторів форм і звітів мають однаковий інтерфейс і надають користувачеві багато можливостей. Форма або звіт конструюється за принципом WYSIWYG (What You See Is What You Get - що бачиш, то й одержиш). Додаючи черговий елемент управління, користувач бачить, як при цьому змінюється створювана форма.

з форми й звіти можна включати написи, поля текстових даних, перемикачі, прапорці, лінії й прямокутники, а також оформляти їх, виділяючи елементи кольором і тінню. Більше того, можна включати цілі малюнки, діаграми,

подформы й підзвіти. При цьому всі параметри подання даних залишаються повністю підконтрольними користувачеві. Форми можуть займати багато сторінок, а у звітах може бути передбачене багато рівнів угруповання даних і підведення підсумків. Форми й звіти можна переглядати в режимі попереднього перегляду, забезпечуючи погляд "з висоти пташиного польоту" шляхом зміни масштабу. У режимі конструювання звіт можна переглядати з фіктивними даними, щоб не чекати обробки великого реального файлу.

Конструктор звітів - дуже потужний засіб, що допускає використання до десяти рівнів угруповання й сортування. Завдяки йому існує можливість створення звітів, що демонструють процентні й підсумкові показники, одержати які можна лише за два проходи. Допускається створення багатьох типів звітів, які включають поштові наклейки й списки розсилання пошти.

Багатотабличні запити й відносини. Одна із самих потужних можливостей Access одночасно є й найбільш важливою. Відносини дозволяють зв'язати таблиці графічно. Можна навіть зв'язувати таблиці, що представляють файли різних типів (наприклад, таблицю Access і таблицю dBASE). Після подібного зв'язування таблиці виступають вже як одне ціле, і тепер можна будувати запити стосовно до будь- яких даних у них. Можна вибирати конкретні поля, визначати порядок сортування, вводити критерії відбору потрібних записів. Можна відображати результати виконання запиту у вигляді таблиці, форми або звіту. Від користувача не потрібно попередньої установки зв'язків: замість цього досить увійти в конструктор запитів (наприклад, коли потрібно побудувати певний звіт).

Запити застосовують і в інших випадках. Можна створювати запити, які забезпечують обчислення підсумків, відображення згрупованих і побудова нових таблиць. Запит можна використовувати навіть для відновлення даних у таблицях, видалення записів і додавання однієї таблиці до іншої.

Графіки й діаграми. В Access використовується той же самий графічний додаток, що й в Microsoft Word, Excel, PowerPoint і Project. Воно дозволяє створювати сотні типів графіків і діаграм, набудовуючи їх, виходячи з конкретних потреб. Можна створювати БДтограми, лінійчаті, кругові , поверхневі й інші

діаграми, причому як двох-, так і тривимірні. Їх можна довільно супроводжувати текстом, оформляти різними квітами й візерунками. Значення можуть відображатися в стовпцях або секторах кругових діаграм. Можна розвертати зображення діаграм так, щоб вони відтворювалися під будь-яким зручним кутом зору. Все це забезпечує програма Access Graph.

Можливості DDE і OLE

За допомогою DDE (Dynamic Data Exchange - динамічний обмін даними) і OLE (Object Linking and Embedding - зв'язування й впровадження об'єктів) у форми й звіти Access можна додавати всілякі нові об'єкти. Такими об'єктами можуть бути звук, малюнки, діаграми й навіть відеокліпи. Можна впроваджувати об'єкти OLE (наприклад, растрові зображення) або документи текстових процесорів (Word або WordPerfect) або встановлювати зв'язку з електронними таблицями Excel. Зв 'язуючи ці об'єкти зі своєю базою даних, користувач може створювати динамічні форми й звіти, а також використовувати ту саму інформацію в різних додатках Windows.

Доступ до Internet. В Access тепер передбачені всі можливості, що забезпечують зв'язок додатку з Internet/intranet. Одним клацанням кнопкою миші можна зберегти таблиці, запити, форми й звіти у форматі HTML. Відповідний майстер дозволяє навіть новачкові перенести коди HTML з об'єкта на Web-Сторінку, роблячи їх доступними для використання всім, хто подорожує по Internet! Гіперпосилання дозволяють одержувати доступ до даних, які розміщені на Web-Сторінці, прямо з форм Access.

Багато хто вважають, що розміщення даних на Web-Сторінках повинне здійснюватися Web-Адміністраторами. Access з повною визначеністю доводить, що ця операція може бути з успіхом виконана будь-яким користувачем. А допоможе йому в цьому майстер розміщення на Web-Сторінці , що забезпечує перетворення обраних об'єктів бази даних у формат HTML і перенос їх уже в такому виді на Web-Сторінку. За допомогою цього майстра можна створити статичні або динамічні сторінки, перенести їх на Web-Сервер, створити свою початкову сторінку й навіть використовувати шаблони для одержання стандартного зовнішнього вигляду всіх HTML-Сторінок!

Вбудовані функції. Access містить понад сто функцій (невеликих вбудованих програм, які в результаті виконання повертають значення), що виконують безліч різноманітних завдань. Є функції для маніпулювання базами даних, рядками, числами у форматі дати й часу, математичні, ділові й фінансові. Їх можна використовувати для створення виражень, що обчислюються, у формах, звітах і запитах.

Макроси: програмування без програмування.

Для непрограмістів (або досвідчених користувачів, які просто не бажають програмувати) в Access передбачені макроси. Вони дозволяють автоматизувати виконання деяких завдань. Біля п'ятдесятьох макросів дають можливість маніпулювати даними, створювати меню й діалогові вікна, відкривати форми й звіти, словом, автоматизувати виконання практично будь-якого завдання. За допомогою макросів можна вирішити порядка 90% всіх завдань обробки даних.

Модулі: Visual Basic for Applications програмування баз даних.

Access - це серйозне середовище розробки додатків з повнофункціональною мовою програмування. Мова VBA (раніше відомий як Access Basic) реалізує об'єктно-орієнтований підхід до програмування й дозволяє програмістові робити практично все, що тільки можна собі представити. Це потужна мова структурного програмування. Він є повністю розширюваним і підтримує процедури API у будь-яких динамічних бібліотеках (DLL) операційних систем Windows.

Повнофункціональне середовище розробки підтримує безліч потужних сучасних можливостей: багатовіконий режим для редагування й налагодження, автоматичну перевірку синтаксису, контрольні крапки, покрокове виконання й навіть синтаксична довідка, що відображає на екрані варіанти команд, що вводяться [23, 28].

5.4 Адміністратор бази даних

Адміністратор даних (АД) — це людина, відповідальна за стратегію й політику прийняття рішень, пов'язаних з даними об'єкта, а адміністратор бази даних (АБД) — це людина, що забезпечує необхідну технічну підтримку виконання цих рішень. Таким чином, АБД відповідає за загальне управління системою на технічному рівні. Функції АБД

у Визначення концептуальної схеми

Адміністратор даних визначає, які саме дані необхідно зберігати в базі даних, тобто визначає об'єкти, у яких зацікавлене підприємство, а також інформацію, яку необхідно записувати про ці об'єкти. Цей процес звичайно називають логічним (або концептуальним) проектуванням бази даних. Після того як вміст бази даних визначено адміністратором даних на абстрактному рівні, АБД створює відповідну концептуальну схему за допомогою концептуальної мови визначення даних. Об'єктна (відкомпільована) форма цієї схеми буде використовуватися СУБД при відповідях на запити, пов'язані з доступом. Вихідна (не відкомпільована) форма буде відігравати роль довідкового документа для користувачів системи.

- Визначення внутрішньої схеми

Адміністратор бази даних повинен також вирішувати, як дані повинні бути представлені в збереженій базі даних. Цей процес звичайно називають фізичним проектуванням бази даних. Після завершення фізичного проектування АБД за допомогою внутрішньої мови визначення даних повинен створити відповідну структуру зберігання (тобто описати внутрішню схему). Крім цього, він повинен визначити відповідне відображення між внутрішньою й концептуальною схемами. На практиці концептуальна й внутрішня мови визначення даних можуть містити в собі засобу визначення цього відображення, але ці дві функції повинні бути чітко розділені. Як і концептуальна схема, внутрішня схема й відповідне відображення будуть існувати у вихідній і об'єктній формах.

Взаємодія з користувачами обов'язки АБД також входить взаємодія з користувачами, забезпечення наявності необхідних їм даних і написання (або надання допомоги користувачеві в написанні) необхідних зовнішніх схем за допомогою прикладної зовнішньої мови визначення даних. Крім цього, необхідно також визначити відображення між кожною зовнішньою й концептуальною схемами. На практиці зовнішня мова визначення даних може включати засоби визначення цього відображення, але знову ж схема відображення повинні бути чітко розділені. Зовнішня схема й відповідне відображення будуть існувати у вихідній і об'єктній формах.

Інші аспекти взаємодії з користувачами включають консультації по розробці додатків, забезпечення технічного утворення, допомога у виявленні й рішенні виникаючих проблем та інше пов'язане із системою професійне обслуговування.

- Визначення правил безпеці й цілісності

Правила безпеки й цілісності розглядаються як частина концептуальної схеми. Концептуальна мова визначення даних повинен містити в собі засоби визначення цих правил.

У Визначення процедур резервного копіювання й відновлення

Після того як об'єкт довірив свої дані системі баз даних, воно стало критично залежним від успішного функціонування системи. У випадку ушкодження якої-небудь частини бази даних внаслідок помилки людини, відмови встаткування або збоїти допоміжної операційної системи дуже важливо мати можливість відновити дані з мінімальною затримкою й з найменшим впливом на іншу частину системи. В ідеалі дані, які не були ушкоджені, зовсім не повинні зачіпатися. Адміністратор бази даних повинен визначити й реалізувати підходящу схему відновлення , що використовує, наприклад, періодичне вивантаження (або "дампинг") бази даних на пристрій резервного копіювання й процедури, при необхідності загружаючи базу даних з останнього дампа.

Управління продуктивністю й реагування на вимоги, що змінюються АБД відповідає за таку організацію системи, при якій можна одержати найбільшу для всього об'єкту продуктивність, а також за перенастроювання системи відповідно до змінюющихся вимог. Наприклад , може виникнути необхідність у поетапній реорганізації збереженої бази даних, щоб бути впевненим, що рівень продуктивності залишається прийнятним. Як уже згадувалося, будь-які зміни на рівні фізичної пам'яті системи (внутрішньому рівні) повинні супроводжуватися відповідними змінами визначень відображення з концептуального рівня, тому концептуальна схема може залишатися незмінною.

Питання для контролю

1. Три основні категорії засобів моделювання даних.

2. Системи управління базами даних.
3. Логічна модель баз даних.
4. Характеристики Access.
5. Поняття адміністратор бази даних.

Лекція 6

Інформаційна модель СУБД

6.1 Попереднє планування, підготовка даних, послідовність створення інформаційної моделі

При проектуванні системи обробки даних найбільш важлива організація даних. Допомогти зрозуміти організацію даних покликана інформаційна модель.

Процес створення інформаційної моделі починається з визначення концептуальних вимог ряду користувачів. Концептуальні вимоги можуть визначатися й для деяких завдань (додатків), які найближчим часом реалізовувати не планується. Це може трохи підвищити трудомісткість роботи, однак допоможе найбільше повно врахувати всі нюанси функціональності, необхідної для розроблювальної системи, і знизить імовірність переробки надалі. Вимоги окремих користувачів повинні бути представлені в єдиному «узагальненому поданні». Останнє називають концептуальною моделлю.

Об'єкт - це абстракція безлічі предметів реального миру, що володіють однаковими характеристиками й законами поводження. Об'єкт являє собою типовий невизначений екземпляр такої безлічі.

Об'єкти поєднуються в класи за загальними характеристиками. Клас - це безліч предметів реального миру, зв'язаних спільністю структури й поводженням.

Концептуальна модель представляє об'єкти і їхні взаємозв'язки без указування способів їхнього фізичного зберігання. Таким чином, концептуальна модель є, по суті, моделлю предметної області. При проектуванні концептуальної моделі всі зусилля розроблювача повинні бути спрямовані в основному на структуризацію даних і виявлення взаємозв'язків між ними без розгляду особливостей реалізації й питань ефективності обробки. Проектування концептуальної моделі засновано на аналізі розв'язуваних на цьому підприємстві завдань по обробці даних. Концептуальна модель включає описи об'єктів і їхніх взаємозв'язків, що представляють інтерес у розглянутій предметній області й

даних, що виявляються в результаті аналізу. Маються на увазі дані, використовувані як у вже розроблених прикладних програмах, так і в тих, які тільки будуть реалізовані.

Нормалізація інформаційної моделі виконується в кілька етапів.

Дані, представлені у вигляді двовимірної таблиці, є **першою** нормальною формою реляційної моделі даних. Перший етап нормалізації полягає в утворенні двовимірної таблиці, що містить всі необхідні властивості інформаційної моделі, і у виділенні ключових властивостей.

Очевидно, що отримана досить значна таблиця буде містити дуже різнорідну інформацію. У цьому випадку будуть спостерігатися аномалії включення, відновлення й видалення даних, тому що при виконанні цих дій нам оведеться приділити увага даним (уводити або піклуватися про те, щоб вони не були стерті), які не мають до поточних дій ніякого відношення. Наприклад, може спостерігатися така парадоксальна ситуація.

Відношення задане в **другій нормальній формі**, якщо воно є відношенням у першій нормальній формі й кожна властивість, що не є первинною властивістю щодо цього, повністю залежить від будь-якого можливого ключа цього відношення.

Якщо всі можливі ключі відносини містять по одній властивості, то це відношення задане в другій нормальній формі, тому що в цьому випадку всі властивості, що не є первинними, повністю залежать від можливих ключів. Якщо ключі складаються більш ніж з однієї властивості, відношення, задане в першій нормальній формі, може не бути відношенням у другій нормальній формі. Приведення відносин до другої нормальної форми полягає в забезпеченні повної функціональної залежності всіх властивостей від ключа за рахунок розбивки таблиці на трохи, у яких всі наявні властивості будуть мати повну функціональну залежність від ключа цієї таблиці. У процесі приведення моделі до другої нормальної форми в основному виключаються аномалії дублювання даних.

Відношення задане в **третій нормальній формі**, якщо воно задано в другій нормальній формі й кожна властивість цього відношення, що не є первинним, не транзитивно залежить від кожного можливого ключа цього відношення.

Транзитивна залежність виявляє дублювання даних в одному відношенні. Якщо A, B і C - три властивості одного відношення й C залежить від B, а C від A, то говорять, що C транзитивно залежить від A. Перетворення в третю нормальну форму відбувається за рахунок поділу вихідного відношення на два.

Концептуальна модель переноситься потім у модель даних, сумісну з обраної СУБД. Можливо, що відбиті в концептуальній моделі взаємозв'язки між об'єктами виявляться згодом нереалізованими засобами обраної СУБД. Це зажадає зміни концептуальної моделі. Версія концептуальної моделі, що може бути забезпечена конкретної СУБД, називається **логічною моделлю.**

Логічна модель відбиває логічні зв'язки між елементами даних поза залежністю від їхнього змісту й середовища зберігання. Логічна модель даних може бути реляційною, ієрархічною або мережною. Користувачам виділяються підмножини цієї логічної моделі, називані зовнішніми моделями, що відбивають їхні подання про предметну область. Зовнішня модель відповідає поданням, які користувачі одержують на основі логічної моделі, у той час як концептуальні вимоги відбивають подання, які користувачі спочатку бажали мати і які лягли в основу розробки концептуальної моделі. Логічна модель відображається у фізичну пам'ять, таку, як диск, стрічка або який-небудь інший носій інформації.

Ієрархічна модель даних будується за принципом ієрархії типів об'єктів , тобто один тип об'єкта є головним, а інші , що перебувають на нижчих рівнях ієрархії, — підлеглими. Між головним і підлеглим об'єктами встановлюється взаємозв'язок «один до багатьох». У той же час для кожного екземпляра головного об'єкта може бути кілька екземплярів підлеглих типів об'єктів. Взаємозв'язку між об'єктами нагадують взаємозв'язки в генеалогічному дереві за єдиним виключенням: для кожного породженого (підлеглого) типу об'єкта може бути тільки один вихідний (головний) тип об'єкта.

Отже, отриману концептуальну модель, будемо вважати логіко-ієрархічною моделлю даних. На мою думку, більше перетворень не вийде. Кінцеву модель можна вважати кінченої.

Фізична модель, що визначає розміщення даних, методи доступу й техніку індексування, називається внутрішньою моделлю системи.

Зовнішні моделі ніяк не пов'язані з типом фізичної пам'яті, у якій будуть зберігатися дані, і з методами доступу до цих даних. Це положення відбиває **перший рівень незалежності даних**. З іншого боку, якщо концептуальна модель здатна враховувати розширення вимог до системи майбутньому, те внесені в неї зміни не повинні робити впливу на існуючі зовнішні моделі. Це — **другий рівень незалежності даних**. Побудова логічної моделі обумовлено вимогами використовуваної СУБД. Тому при заміні СУБД вона також може змінитися.

погляду прикладного програмування незалежність даних визначається не технікою програмування, а його дисципліною, тобто для того щоб при будь-якій зміні системи уникнути перекомпіляції додатка, рекомендується не визначати константи (постійні значення даних) у програмі. Краще рішення складається в передачі програмі значень як параметри.

Всі актуальні вимоги предметної області й адекватні їм «сховані» вимоги на стадії проектування повинні знайти своє відбиття в концептуальній моделі. Звичайно, не можна передбачити всі можливі варіанти використання й зміни бази даних. Але в більшості предметних областей такі основні дані, як об'єкти і їхні взаємозв'язки, відносно стабільні. Міняються тільки інформаційні вимоги, тобто способи використання даних для одержання інформації.

Ступінь незалежності даних визначається старанністю проектування бази даних. Всебічний аналіз об'єктів предметної області і їхніх взаємозв'язків мінімізує вплив зміни вимог до даних в одній програмі на інші програми. У цьому й складається всеосяжна незалежність даних.

Основне розходження між зазначеними вище трьома типами моделей даних (концептуальною, логічною й фізичною) складається **в способах подання взаємозв'язків між об'єктами**. При проектуванні БД потрібно

розрізняти взаємозв'язку між об'єктами, між властивостями одного об'єкта з між властивостями різних об'єктів.

процесі проектування об'єкти перетворяться у відносини, властивості в поля таблиць, методи - у процедури, форми й т.д. Правильно проведений об'єктно-орієнтований аналіз дозволяє значно полегшити роботу.

6.2 Електронні таблиці.

Електронні таблиці, також як файли баз даних формату .DBF, виникли як якась метафора таблиці. Проста ідея про те, що було б добре, якби в деяких клітках таблиці відразу ж відображався результат обчислень, що залежить від даних в інших клітках таблиці, привела до появи в 1979 році програми VisiCalc. У лютому 1981 року з 'явилася програма SuperCalc, а на початку 1983 року - пакет Lotus 1-2-3. В останній із цих програм було передбачене відображення результатів розрахунків у графічній формі й це багато в чому визначило успіх Lotus 1-2-3 на ринку в середині 80-х років. Наприкінці 80-х років ситуація на ринку електронних таблиць змінилася - завзята конкурентна боротьба йшла між програмами 1-2-3 версії 2.01 і 3.0 фірми Lotus Development Corporation, Excel 2.1 фірми Microsoft, Quattro 1.0 фірми Borland International і SuperCalc 5 фірми Computer Associates. До середини 1990 років найбільшу популярність придбав пакет Excel, чому чимало сприяли ринкові успіхи системи Microsoft Windows.

Електронні таблиці Excel дозволяють вирішувати дуже багато завдань обробки даних, сформованих у таблиці . В Excel передбачений автоматичне уведення даних з SQL-Серверів і різних СУБД. Обробка даних передбачає, крім арифметичних операцій, можливість маніпулювання комплексними числами й матрицями, обчислення параметрів регресійних залежностей, перетворень Фур 'є й інших статистичних характеристик. Передбачені дуже різноманітні методи для графічного висновку результатів, у тому числі, починаючи з Excel 7.0, відображення одержуваних результатів на числові карти у форматі БД MapInfo. Складні завдання можуть вирішуватися в межах декількох зв'язаних таблиць, мають назву в Excel’і робочою книгою.

8. пакеті Excel передбачені різні засоби для збереження логічної цілісності інформації - автоматична заміна номерів аркушів і осередків при внесенні змін у книгу, автоматичне копіювання формул зі зрушенням номерів осередків у межах стовпців з подібними обчисленнями й багато чого іншого.

Питання для контролю.

1. Принципи логічного і фізичного проектування БД.
2. Корпоративні та «настільні» СУБД.
3. Переваги й недоліки корпоративних та «настільних» СУБД.
4. Послідовність створення інформаційної моделі.
5. Електроні таблиці.

Лекція 7

МОВА SQL. ОПЕРАТОРИ ОПИСУ СХЕМ БД

7.1 Структурована мова SQL. Особливості та визначення

В кінці 1970-х групою дослідників із IBM була розроблена у рамках проекту експериментальної реляційної СКБД System R. Для отримання і маніпуляції з даними в System R була включена мова SEQUEL (Structered English QUEry Language – структурована англійська мова запитів). Пізніше абревіатура SEQUEL була скорочена до SQL, оскільки вияснилося, що SEQUEL є торговою маркою, зарегістрованою авіакомпанією Hawker-Siddeley у Великобританії.

Американський інститут національних стандартів (ANSI) та міжнародна організація стандартів (ISO) займаються описом і підтримкою стандартів мови SQL. Усі сучасні СКБД підтримують певний стандарт, проте є й відхилення, які в кожному конкретному випадку специфікуються в документації програмного продукту.

SQL (Structered Query Language – структурована мова запитів) – стандартна мова, яка використовується для взаємодії з реляційною базою даних. SQL – найбільш поширена мова для написання запитів до баз даних. SQL не є процедурною мовою, як Pascal, Basic, FORTRAN, COBOL, Ada. Багато постачальників СКБД використовують процедурні розширення мови SQL, такі як OraclebPL/SQL (Procedural Language – процедурна мова), чи Microsoft Transact – SQL, але всі ці розширення SQL насправді є новими мовами – використовуваний в них SQL не є процедурним. SQL також не потрібно путати з об’єктно – орієнтованими мовами, такими як Java чи C++.

SQL – мова для управління реляційними базами даних, часто використовується в поєднанні з процедурними чи об’єктно-орієнтованими мовами.

Формат запису операторів SQL вільний. Можна писати усе підряд в одному рядку, один оператор на кількох рядках, слова операторів можна розділяти довільною кількістю пробілів і коментарів.

Обмеження щодо довжини назв змінних: ім’я бази даних має містити не більше 10 символів, імена інших об’єктів SQL (таблиць, стовпців, псевдо таблиць (віртуальних) – не більше 18 символів.

Синтаксис SQL:

- Кожен запит починається з команди.
- Кожен запит завершується розділювачем, як правило – крапка з комою (деякі реалізації, наприклад, Oracle не будуть виконувати запит без завершуючого роздільника, інші – вважають його необов’язковим).
- Елементи мови розділяються одним чи кількома пробілами.
- Запити пишуться у вільній формі, тобто не існує строгих правил відносно положення елементів мови в стрічці чи положення роздільника стрічок.
- Елементи мови SQL можна писати як в верхньому, так і в нижньому і змішаному регістрах. Але в більшості реалізаціях і у відповідності з стандартами ANSI/ISO всі команди і імена об’єктів бази даних (таблиці, поля і т. д.) повинні бути у верхньому регістрі. Виключення становлять Microsoft SQL Server і Sybase, для яких імена об’єктів, написаних в різному регістрі, обробляються як різні імена.
- Для розділення елементів списку використовується кома (наприклад, іменастовпчиків). Пробіли після кожної коми необов’язкові.
- Текстові стрічки в SQL-запитах пишуться в кавичках. Числові константи в кавички не заключаються.
- Імена об’єктів бази даних можуть включати букви, цифри і символ підкреслення.
- Однострічковий коментар починається з двох дефісів підряд (--), які можуть знаходитися на початку стрічки – в цьому випадку вся стрічка стає коментарем, чи в будь-якій частині стрічки – тоді частина стрічки сприймається як коментар.
- Багатострічковий коментар починається в поєднанні з /* і продовжується, поки не зустрінеться обернена комбінація */.

7. 2 Складові частини SQL

Всі команди мови SQL за функціями можна поділити на такі групи:

Мова опису даних (Data Definition Language – DDL). Включає SQL-запити, які дозволяють створювати і модифікувати структуру об’єктів бази (таблиці, представлення, індекси).

Мова запиту даних (Data Query Language – DQL). Ця мова використовується для отримання даних із бази. Головною інструкцією цієї мови є команда SELECT. Іноді оператор DQL відносять до мови маніпулювання даними.

Мова маніпулювання даними (Data Manipulation Language – DML). До неї відносяться команди INSERT і DELETE, які дозволяють додавати і вилучати рядки з таблиць, а також команда UPDATE, яка вносить зміни окремих значень полів (модифікує наявні дані в базі).

Мова обробки транзакцій (Transaction Processing Language – TPL). Включає команди DEGIN TRANS[ACTION], COMMIT [WORK], ROOLBACK [WORK], які об'єднують кілька команд DML. Якщо при виконанні транзакції пройшов збій, то попередні операції відміняються і виконується відкат трансакції.

Мова управління курсором (Cursor Control Language – CCL). Команди цієї мови дозволяють виділити для обробки один рядок результатного набору записів.

Мова управління даними (Data Control Language – DCL). Забезпечує виконання адміністративних функцій надання та відміни прав доступу до всієї бази даних (команди GRANT REVOKE), набору таблиць чи спеціальним командам SQL.

Усі ключові слова можна, в свою чергу, поділити на такі категорії:

Команди. Це дієслова, які визначають дії, котрі необхідно виконати, наприклад, SELECT.

Умови. Обмежують діапазон значень елементів, що входять до запиту, наприклад, WHERE.

Модифікатори. Змінюють виконання команди, наприклад, ORDER BY.

Оператори. Порівнюють значення полів і застосовуються в умовах відбору записів, а також для створення міжтабличних зв'язків.

Статистичні (агрегатні) функції. Повертають одне результатне значення, яке обчислюється для певного набору даних.

Інші ключові слова, що змінюють дії команд чи управляють Курсором (вказівником поточного запису), за допомогою якого вибираються окремі рядки запиту.

7.3 Оператори опису схем БД

Мова опису даних DDL

Мова DDL є підмножиною мови SQL, за допомогою якої можна створювати та вилучати об'єкти бази даних.

У SQL розрізняються наступні види об'єктів:

- база даних (database);
- таблиця (table);
- стовпець (column);
- індекс (index).
- віртуальна таблиця (view);

Кожен об'єкт має власне ім'я – ідентифікатор, а також власника – користувача, що його створив. Якщо в якомусь запиті чи операторі SQL згадуються два поля з однаковими назвами, то їх потрібно уточнювати повними іменами таблиць, що їх містять. Перед ім'ям будь-якого об'єкта можна зазначати ім’я його власника (owner-name) – вхідне ім’я користувача, який створив цей об’єкт.

Приклади уточнення імені поля

Студенти.Прізвище	Поле *Прізвище*з таблиці *Студенти*
[Нові студенти].Прізвище	Поле *Прізвище*з таблиці*Нові Студенти*
Петренко.Студенти.Прізвище	Поле *Прізвище*з таблиці *Студенти*, власником

	якої є Петренко

Основні команди DDL:

CREATE – створює новий об’єкт бази даних.

ALTER – змінює опис існуючого об’єкта бази даних.

DROP – видаляє існуючий об’єкт бази даних.

Оператор **CREATE DATABASE** – використовується для створення нової бази даних.

CREATE DATABASE ім’я_бази [особливі параметри постачальника]

Приклад:

CREATE DATABASE Студенти

Оператор **DATABASE** відкриває нову базу, закриваючи при цьому доступ до об'єктів попередньої поточної (CURRENT) бази.

Для переключення до нової бази даних, наприклад, Base1 слід вказати:

DATABASE Base1 - поточною є база Base1.

Оператор **CLOSE DATABASE** закриває поточну базу даних.

CLOSE DATABASE - поточної бази немає.

Створення таблиці.

CREATE TABLE ім’я_таблиці

(<визначення поля>

[,<визначення поля>]

[,<обмеження таблиці>]…);

Оператор **CREATE TABLE** складається в основному з переліку імен стовпців таблиці із наданням наступної інформації для кожного стовпця:

ім'я поля;

тип даних – категорія формату конкретного поля; кожна система має свої типи даних;

обмеження PRIMARY KEY, якщо стовпець (або стовпці) є первинним ключем;

обмеження FOREIGN KEY, якщо стовпець (або стовпці) є зовнішнім ключем;

обмеження NOT NULL, якщо стовпець не повинен містити NULL-значення;

обмеження UNIQUE, якщо в стовпці не дозволена наявність дублікатів;

обмеження DEFAULT, якщо стовпець має значення за умовчанням;

обмеження CHECK, якщо для стовпця існують значення для перевірки.

Обмеження таблиці може накладатись більше ніж на одне поле.

Приклад Створити таблицю *My* і провести індексацію по ключовому полю *ID*

CREATE TABLE My (

[ID] Counter,

[Вік] INT,

[прізвище] CHAR(20) UNIQUE,

[ім’я] CHAR(15),

[стипендія] MONEY,

[код] LONG,

[ріст] FLOAT,

[вага] REAL,

[курс] BYTE,

[дата народження] DATE,

[дата поїздки] DATETIME,

[час початку] TIME,

[клуб] LOGICAL,

[фото] IMAGE,

[адреса] TEXT,

[примітки]MEMO,

CONSTRAINT [Index1] PRIMARY KEY ([ID]))

Створення таблиці на базі існуючої.

Для створення таблиць найчастіше використовують команду CREATE TABLE. Однак деякі системи надають альтернативний спосіб означення таблиць із використанням формату та даних наявної таблиці. Цей метод корисний для вибирання даних із таблиці з метою тимчасового зберігання та модифікації.

CREATE TABLE ім'я_таблиці

(<визначення поля>

[,<визначення поля>]…) AS (SELECT<визначення поля> FROM ім'я_існуючої_таблиці [WHERE…]);

Дана команда дає можливість створити нову таблицю з типами даних полів наявної таблиці та за необхідності перейменувати поля.

Оператор **CREATE INDEX** створює новий індекс для існуючої таблиці.

CREATE [UNIQUE] INDEX ім'я_індексу ON ім'я_таблиці

(ім'я_поля [ASC | DESC] [,ім'я_поля [ASC | DESC]…]);

Необов'язковий параметр UNIQUE визначає індекс, як унікальний, при якому ніякі дві стрічки в таблиці не можуть мати однакові значення.

Необов'язкове ключове слово ASC створює індекс у зростаючому порядку, DESC – в спадаючому. Якщо ні одне з них не вказано, по замовчуванню вибирається зростаючий порядок.

Індекс повинен включати хоча б одне поле. Верхнього обмеження на кількість полів практично не існує.

Приклад. Створити індекс *index2* для сортування таблиці *My* за полем прізвище.

```
CREATE UNIQUE INDEX index2 ON My (прізвище)
```

Оператор **ALTER TABLE** – використовується для внесення змін до існуючої таблиці. За допомогою цього оператора можна поміняти тип поля, додавати нові поля в таблицю чи вилучати існуючі;

```
ALTER TABLE ім'я_таблиці
ADD (<визначення поля>
[,<визначення поля>]…);
```

Приклад. Додати до таблиці поле *Кафедра*, яке містить текстові значення довжиною до 20 символів:

```
ALTER TABLE my
ADD [кафедра] CHAR(20)
```

Приклад. Видалити поля *Вік* та *Кафедра.*

```
ALTER TABLE my
DROP [ВІК], [кафедра]
```

Зміна структури таблиці призводить до фізичного перетворення даних у ній. Якщо змінений тип стовпці, то дані в ньому перетворяться до нового типу;

Якщо це неможливо здійснити, то оператор ALTER видає помилку, а таблиця залишається у незмінному стані.

7.4 Використання віртуальних таблиць

Віртуальна таблиця – це пойменована таблиця, одержана в результаті виконання оператора SELECT з можливою заміною імен стовпців.

Віртуальна таблиця застосовується для створення складних запитів. Вона може використовуватися в операторах SELECT, INSERT, INPUT, UPDATE, DELETE, хоча фізично не займає місця в базі даних, як звичайна таблиця. Синтаксис створення віртуальної таблиці такий:

```
CREATE VIEW ім’я_віртуальної_таблиці [(<список полів>)] AS
SELECT <список полів>
FROM  <імена таблиць>
[WHERE ...]
[WITH CHECK OPTION]
```

Необов'язкова фраза WITH CHECK OPTION (з контролем) вказує на те, що для операцій INSERT та UPDATE над цією таблицею має здійснюватися контроль, який забезпечує виконання умов, заданих у фразі WHERE підзапиту.

Проста віртуальна таблиця – це віртуальна таблиця, що є копією вихідної, але має інше ім'я:

```
CREATE VIEW  КОПІЯ_ФАКУЛЬТЕТУ AS
SELECT  *
FROM  ФАКУЛЬТЕТ
```

У результаті виконання наведеного оператора створюється віртуальна таблиця КОПІЯ_ФАКУЛЬТЕТУ, що є точною копією таблиці ФАКУЛЬТЕТ. Можна також створювати віртуальні таблиці на базі інших віртуальних таблиць.

Вибирання стовпців

До віртуальної таблиці можна копіювати окремі стовпці вихідної таблиці:

```
CREATE VIEW ДЕКАНИ_ФАКУЛЬТЕТІВ (Назва. Декан) AS
SELECT  *
FROM  ФАКУЛЬТЕТ
```

Оператор **DROP** вилучає існуючу таблицю з бази даних або вилучає існуючий індекс з таблиці:

DROP <тип_об'єкта> ім'я_об'єкта [<параметри видалення>]

тип_об'єкта вказує тип видаляючого об'єкта, наприклад, INDEX, TABLE, VIEW.

Перш, ніж вилучити таблицю чи вилучити з неї індекс, необхідно її закрити. Також для вилучення індексу з таблиці можна використати ALTER TABLE.

Приклад. Знищити таблиці *My* і *x*.

DROP TABLE my, x

Приклад. Знищити базу даних.

DROP DATABASE Кафедра -- знищує базу разом з усіма даними й системним журналом

Питання для контролю

1. Які оператори мови опису структури таблиці DDL.
2. Для створення віртуальної таблиці використовують команди.
3. Оператори роботи з рядками DML.
4. Оператори роботи з правами доступу DCL.
5. Що таке транзакція, команди роботи з ними.

Лекція 8

ВИБІРКА ДАНИХ ЗА ДОПОМОГОЮ МОВИ ЗАПИТІВ SQL

8.1 Оператор вибору SELECT

Мова запитів даних SQL (DQL) включає лише одну команду **SELECT**.

Оператор SELECT використовується для отримання даних з бази (без їх зміни). У операторі SELECT обов'язковими є два ключових слова – SELECT і FROM. Після першого слідує список полів, що вибираються запитом; після другого вказується перелік таблиць, що беруть участь в запиті. Ключове слово WHERE є необов'язковим. За допомогою оператора SELECT можна виконувати три типи операцій реляційної алгебри: селекція, проекція і з'єднання.

Додаткові параметри оператора SELECT

ALL	Вибирає всі записи, що задовольняють умову в інструкції SQL. Цей аргумент використовується, якщо не вказано інших параметрів (за замовчуванням).
DISTINCT	Вибирає лише ті записи, всі значення яких у полях, перерахованих в інструкції SELECT, є унікальними. Комбінація всіх обраних полів має бути унікальною.
DISTINCTROW	Пропускає дублюючі записи, в яких співпадають не одне поле, а всі поля. Впливає на результат, коли до запиту включені не всі поля з таблиць, що аналізуються.
TOP n	Повертає визначене число записів з початку до кінця впорядкованого

	списку. Замість n підставляється необхідне число записів.
TOP n PERCENT	Повертає визначений відсоток записів з початку до кінця впорядкованого списку. Замість n підставляється необхідний відсоток записів.
*	Вказує, що вибрано всі поля заданої таблиці (таблиць).

Приклад. Вибрати всі рядки і всі стовпчики з таблиці *Студенти.*

```
SELECT *
FROM Студенти;
```

Приклад. Знайти у таблиці *Студенти* рядки, у якому стовпець *Курс* містить число 3. З цього рядка беруться лише три вказані стовпці.

```
SELECT Прізвище, Ім’я, Місто
FROM Студенти;
WHERE Курс = 3
```

Приклад. Вибрати прізвища студентів, міста їхнього проживання з таблиці *Студенти*, а прізвища наукових керівників – з таблиці *Курсові роботи.* Квадратні дужки використовуються для позначення назв полів, що складаються з кількох слів, чи містять у назві нестандартні символи.

```
SELECT Студенти.Прізвище, Студенти.[Ім’я], Студенти.Місто, [Курсові роботи].Керівник
FROM Студенти, [Курсові роботи]
WHERE Студенти.ID=[Курсові роботи].ID;
```

SQL дозволяє створювати нові поля, які комбінують інформацію з інших полів. Якщо в результаті виконання потрібне інше ім’я, то використовується зарезервоване слово AS.

Приклад

```
SELECT [Дата народження]
AS Народження
FROM Студенти;
```

Приклад

```
SELECT COUNT (ID)
AS Кількість
FROM Студенти;
```

8.2 Використання специфікатора WHERE

Специфікатор WHERE використовується, коли потрібно задати умови на записи, які слід опрацювати. Якщо не задавати WHERE, то результатом запиту будуть всі рядки таблиці. Якщо в запиті беруть участь кілька таблиць і не задано WHERE, то результатом виконання буде скалярний добуток таблиць.

У пропозиції WHERE пишеться логічна умова, що отримується за допомогою логічних операторів AND, OR, NOT. WHERE може вміщувати до 40 виразів, зв’язаних логічними операторами

вираз1<вираз2

вираз1>=вираз2 і т. п.

ім’я_поля IS [NOT] NULL

вираз [NOT] BETWEEN вираз1 AND вираз2

вираз [NOT] IN (вираз1,…[,…])

При використанні операторів порівняння для символьних виразів, мається на увазі місцезнаходження символів в таблиці ASCII.

У мові SQL реалізована можливість пошуку в символьних полях по контексту. Це здійснюється за допомогою специфікаторів MATCHES або LIKE. Ключове слово LIKE рекомендоване в стандарті SQL ANSI, а слово MATCHES введено в розширенні SQL INFORMIX. Після слів LIKE і MATCHES слідує вираз, що визначає умову пошуку. У цих виразах можуть використовуватися наступні шаблони. У операторі LIKE використовуються символи:

* – замінює довільну послідовність символів;

? – замінює будь-який одиночний символ.

Приклад Вибрати з таблиці *Студенти* всі прізвища, що містять букву “і” та передостання літера в яких “к”

```
SELECT *
FROM Студенти
WHERE (((Студенти.Прізвище) LIKE “*і*к?”));
```

Оператор SELECT, вкладений в специфікатор WHERE іншого оператора SELECT, називається підзапитом.

Умови з підзапитом застосовуються для:

- порівняння виразу з результатом іншого SELECT оператора;
- визначення належності виразу результатам іншого SELECT оператора;
- зв’ясування порожнистості множини іншого SELECT оператора.

Приклад. Створити під запит, який повертає єдине значення – максимальний розмір стипендії, зовнішній SELECT оператор знаходить прізвища її власників.

```
SELECT Прізвище
FROM Студенти
WHERE Стипендія=( SELECT MAX(Стипендія ) FROM Студенти);
```

Приклад Вивести інформацію про нових студентів із тих міст, представники яких входять до клубу

```
SELECT *
FROM [Нові Студенти]
WHERE Прізвище IS NOT NULL AND Місто IN (SELECT Місто FROM Студенти
WHERE [Учасник клубу]=True;
```

Приклад Вивести дані про студентів, які є учасниками клубу і отримують стипендію щонайменше вдвічі більшу за найбіднішого учасника клубу.

```
SELECT *
FROM Студенти
WHERE ([Учасник клубу]) AND (Стипендія>=(SELECT 2*MIN(Стипендія)
FROM Студенти WHERE [Учасник клубу]=True));
```

8.3 Використання функцій

У запиті SELECT в склад виразів можуть входити одна або декілька функцій. Існують вбудовані функції, функції набору і функції часу.

Функції COUNT(*) – повертає кількість записів, що задовольняють умову, AVG(стовпець), MAX(стовпець), MIN(стовпець), SUM(стовпець), STDEV(стовпець) – повертає середньоквадратичне відхилення у стовпці, VAR(стовпець) – дисперсія у стовпці, називаються функціями набору і використовуються для отримання зведеної інформації по групі рядків.

Приклад. Знайтисередню стипендію студентів 2-го курсу

```
SELECT AVG(Стипендія)
FROM Студенти
WHERE Курс=2;
```

Приклад.Знайти кількість студентів 3-го курсу і помістити отримане число у нове поле *Кількість*

```
SELECT COUNT(*) AS Кількість
FROM Студенти
WHERE Курс=3;
```

Крім функцій набору можна використати тимчасові функції DAY, MDAY, MONTH, WEEKDAY, YEAR, DATE, як в операторі SELECT, так і в специфікаторі WHERE. Ці функції повертають значення, що відповідають виразам, або аргументам, котрі стоять у викликах цих функцій.

Для видачі результатів запиту у відсортованому по якому-небудь стовпцю вигляді, необхідно використати ключове слово **ORDER BY**. За замовчуванням, для числових даних здійснюється сортування в порядку зростання, а для символьних даних впорядкування буде здійснюватися по буквах алфавіту.

Якщо необхідно здійснити сортування по спаданню, то додається ключове слово DESC:

[ORDER BY ім’я_поля1 [ASC|DESC][, ім’я_поля2 [ASC|DESC]][,…]]]

Приклад. Відсортувати записи таблиці *Студенти* спочатку за полем *Прізвище*, а якщо є однакові прізвища – за *ім’ям*.

```
SELECT Прізвище
FROM Студенти
ORDER BY Прізвище, Ім’я;
```

У операторі ORDER BY замість імен стовпців можна вказувати їх порядковий номер у списку вибірки.

8.4 Специфікатор GROUP BY

Специфікатор GROUP BY використовується при формуванні підсумкових запитів. Об'єднує записи з однаковими значеннями у вказаному списку полів в один запис і видає результуючий запис.

Приклад. Вивести кількість студентів кожного курсу, середню стипендію на кожному курсі з середньоквадратичним відхиленням.

```
SELECT Курс, COUNT(*) AS Кількість, AVG(Стипендія) AS [Середня стипендія],
STDEV(Стипендія) AS Відхилення
FROM Студенти
GROUP BY Курс;
```

8.5 Специфікатор HAVING

Специфікатор HAVING доповнює GROUP BY за допомогою уточнюючих умов до груп рядків, після їх формування. Після того, як записи будуть згруповані за допомогою GROUP BY,HAVING вказує, які з отриманих записів повинні бути відібраними.HAVINGможе містити до 40 виразів, зв'язаних логічними операторами.

Приклад. Вивести назви міст і середні значення стипендії студентів, якщо місто представлено більше, ніж двома студентами.

```
SELECT Місто, Avg(Стипендія) AS [Середня стипендія]
FROM Студенти
GROUPBY МістоHAVING (((Count(*))>2));
```

Оператор SELECT...INTO створює запит на створення нової таблиці.

```
SELECT ім'я_поля1 [,ім'я_поля2[,…]] INTO нова_таблиця [IN
зовнішня_база_даних]
FROM джерело
```

ім'я_поля1, ім'я_поля2 – імена полів, які потрібно скопіювати в нову таблицю;

Приклад. Створити нову таблицю *x*, в яку виводяться лише прізвища всіх студентів із таблиці *Студенти*.

```
SELECT Студенти.Прізвище INTO x
FROM Студенти;
```

Мова обробки транзакцій

Транзакція – це сукупність операцій з маніпулювання базою даних, що має розглядатися як атомарна дія. Під терміном «атомарна дія» мається на увазі така дія, що повністю виконується або скасовується як єдине ціле, тобто щодо неї підтримується принцип «усе або нічого». Різні СКБД мають свою специфіку щодо підтримки транзакцій. Тому використовуватимемо узагальненийсинтаксис таких операцій.

Транзакція активізується командою

BEGIN TRANSACTION [<ім'я транзакції>]

і завершується командою

COMMIT [TRANSACTION]

за якою відбувається фактична зміна стану бази даних згідно з командами, і містяться в транзакції. Наприклад, якщо є така транзакція:

BEGIN TRANSACTION

INSERT INTO ФАКУЛЬТЕТ VALUES (1, "інформатики". "Іванов". "2/b". 42000)

INSERT INTO ФАКУЛЬТЕТ VALUES (1, "економіки", "Петров", "3/a", 47000)

COMMIT

то в таблицю ФАКУЛЬТЕТ будуть вставлені або два рядки, або жодного – залежно від успішності виконання всієї транзакції.

Скасування транзакції.

Після здійснення транзакції надається можливість з'ясувати, чи успішним було її виконання. Транзакцію можна скасувати навіть у разі її успішного виконання, застосувавши команду ROLLBACK (але вона має бути виконана до команди COMMIT). У цьому випадку база даних набуває того ж вигляду, який вона мала виконання транзакції. Можна також вказати так звані точки збереження (save- point), або контрольні точки.

Синтаксис команди ROLLBACK:

ROLLBACK [TRANSACTION] [TO SAVEPOINT <ім'я точки збереження>]

Приклад.

```
BEGIN TRANSACTION
INSERT INTO  ФАКУЛЬТЕТ VALUES (1,   "інформатики",
"Іванов", "2/b",  42000)
ROLLBACK TRANSACTION
INSERT INTO  ФАКУЛЬТЕТ VALUES  (1, "економіки", "Петров",  "3/a",  47000)
COMMIT
```

У таблицю ФАКУЛЬТЕТ буде вставлено лише відомості про факультет економіки, оскільки після додавання першого рядка відбувається скасування транзакції (додавання першого рядка анулюється).

Після застосування команди COMMIT усі дії, що є в транзакції, виконуються. Відміна транзакції призводить до скасування всіх дій, виконаних до моменту застосування команди ROLLBACK TRANSACTION.

За допомогою *точок збереження* можна скасовувати не всі виконані з початку транзакції дії, а лише їх частину.

Синтаксис оголошення точки:

SAVEPOINT <ім'я точки збереження>

Приклад.

```
BEGIN TRANSACTION
UPDATE КАФЕДРА
SET Фонд = Фонд + Фонд/10 WHERE Корпус = "2/3"
SAVEPOINT save_it
DELETE FROM КАФЕДРА WHERE Корпус = "2/3"
ROLLBACK TO SAVEPOINT save_it
COMMIT
```

Видалення рядків не відбудеться, оскільки здійснюється звернення до точки збереження save_it, що означена до команди видалення

Тригер — це SQL-оператор, що активізується під час виконання певних операцій над об'єктами бази даних. Об'єктами бази даних є таблиці, а операціями — додавання, видалення та заміна рядків. Тригери — це один із механізмів підтримки цілісності бази даних.

У найпростішому випадку синтаксис оголошення тригера є таким:

CREATE TRIGGER <ім'я тригера>

{BEFORE | AFTER} <операції над таблицею> [OF <список полів>] ON <ім'я таблиці>

[WHEN (<умова>)]

<оператори SQL>

Якщо умова у фразі WHEN є істинною або ця фраза відсутня, до (BEFORE) або після (AFTER) виконання операції INSERT, UPDATE чи DELETE над таблицею, зазначеною після слова ON, буде виконано вказані нижче оператори SQL. Коли таких операторів кілька, їх слід помістити між ключовими словами BEGIN ATOMIC та END. Конструкція другого рядка означення тригера називається реченням ініціювання, WHEN — умовою ініціювання, а <оператори SQL> - дією тригера.

Приклад.

Після видалення інформації про кафедру видалити інформацію про всіх викладачів кафедри.

CREATE TRIGGER Кафедра_Видалення

AFTER DELETE ON КАФЕДРА

DELETE FROM ВИКЛАДАЧ

WHERE ВИКЛАДАЧ ID = КАФЕДРА ID

Приклад. Після видалення рядка з відомостями про викладача встановити значення NULL в полі ІКуратор тих записів із таблиці ГРУПА, що відповідають групам, де згадани кладач був куратором.

```
CREATE TRIGGER   Викладач_Видалення

AFTER DELETE ON ВИКЛАДАЧ

UPDATE ГРУПА

SET ІКуратор = NULL

WHERE  ГРУПА.IT = ВИКЛАДАЧ.IT
```

Доступ до старих і нових значень рядків

Якщо виконується оновлення рядків таблиці, то в тригері допускається звернення до старих і нових значень рядків, що оновлюються. Для звернення до нового (старого) рядка слід вказати кваліфікатор NEW (OLD) перед іменем стовпця, кваліфікатори дозволяється використовувати як в умові тригера, так і в описі його дії.

Приклад. Після додавання чи оновлення рядка в таблиці КАФЕДРА встановити значення Фонд у таблиці ФАКУЛЬТЕТ рівним сумі фондів усіх кафедр відповідного факультету.

```
CREATE TRIGGER ФАКУЛЬТЕТ_Фонд

AFTER INSERT OR UPDATE ON  КАФЕДРА

BEGIN

UPDATE ФАКУЛЬТЕТ

SET     Фонд = SELECT SUM(Фонд)

FROM КАФЕДРА

WHERE  КАФЕДРА.IF = КАФЕДРА.NEW.IF AND

ФАКУЛЬТЕТ.IF = КАФЕДРА.NEW.IF

END
```

Тригери і транзакції

Дії, які виконуються під час основної операції та в тригері, становлять єдину транзакцію.

1. Перед виконанням додавання, оновлення та видалення неявно ініціюється команда BEGIN TRANSACTION.

2. Реалізується операція додавання/оновлення/видалення.

3. Ініціюється та виконується тригер;

4.Тригер скасовує транзакцію або за замовчуванням його дія завершується.

Приклад.

Дозволити додавання рядка з відомостями про кафедру лише в тому випадку коли існує рядок із даними про факультет, якому належить кафедра (кафедри без факультету не буває).

```
CREATE TRIGGER  Кафедра_Додаааання
BEFORE INSERT ON КАФЕДРА
WHEN NOT EXISTS (SELECT  *
FROM ФАКУЛЬТЕТ
WHERE ФАКУЛЬТЕТ.IF = Кафедра.IF)
BEGIN
ROLLBACK TRANSACTION
END.
```

Питання для контролю

1. Оператор вибору Select з рызними умовами пошуку.
2. Основні характеристики оператора Where.
3. Особливості використання агрегатних функцій.
4. Використання групових операцій.
5. Приклад застосування Having

Лекція 9

МОВА МАНІПУЛЮВАННЯ ДАНИМИ DML

9.1 Оператор DELETE

Мова DML є частиною SQL, призначена для підтримки інформації, що зберігається в реляційних таблицях баз даних. Кожен DML вираз діє на дані лише однієї таблиці. Застосування операторів DELETE, INSERT, UPDATE може привести до порушення цілісності бази даних.

Оператор **DELETE** видаляє записи, які задовольняють певну умову.

```
DELETE
FROM ім'я_таблиці
[WHERE умова];
```

Якщо WHERE відсутнє, то будуть видалені всі стрічки таблиці.

Приклад. Видалити з таблиці *Студенти* всі рядки, у яких номер курсу рівний 5.

```
DELETE
FROM Студенти
WHERE Курс=5;
```

Приклад. Видалити всі рядки з таблиці *Студенти*, але не саму таблицю.

```
DELETE
FROM Студенти
```

9.2 Оператор INSERT INTO

Оператор **INSERTINTO** додає до таблиці одну чи кілька стрічок. Записи додаються в кінець таблиці.

Запит на додавання одного запису:

INSERT INTO ім’я_таблиці

[(список полів)]

VALUES (список_значень);

Якщо відсутній список полів, то потрібно поставити значення для кожного поля таблиці в тому ж порядку, в якому вони знаходяться в таблиці. Рекомендується записувати список полів, інакше вираз INSERT стає залежним від складу таблиці. Тобто, при кожній зміні порядку стовпців чи додаванні нового стовпця запит INSERT під час нового запуску не спрацює. Якщо є список полів, то список значень повинен відповідати значенням стовпців в цьому списку в тому ж порядку. В списку значень можна використовувати ключове слово NULL.

Приклад. Вставити до таблиці *Студенти* запис про нового студента.

INSERT INTO Студенти

VALUES (6, “Петровський”, “Іван” “10.10.1986”, “Київ”, 3, 90, 0);

Якщо використовувати складений вираз, то оператор INSERT INTO може вставити цілий набір, обраних підзапитом SELECT з іншої таблиці записів.

INSERT INTO ім’я_таблиці [IN зовнішня_база_даних]

[(список полів)]

SELECT [джерело.] (список полів)

FROM вираз;

вираз – імена таблиці чи таблиць, звідки вставляються дані. Цей вираз може бути ім’ям окремої таблиці або результатом операції INNER JOIN, LEFT JOIN, RIGHT JOIN, а також збереженим запитом.

Приклад. Додати всі непусті записи з таблиці *Нові студенти* до таблиці *Студенти*. Новий студент буде вчитись на першому курсі, отримувати стипендію 85 грн, не відвідуючи клуб.

INSERT INTO Студенти ([Прізвище], [Ім’я], [Дата народження], [Місто], [Курс], [Стипендія], [Учасник клубу])

```
SELECT [Прізвище], [Ім’я], [Дата народження], [Місто], 1, 85, 0
FROM [Нові студенти]
WHERE [Прізвище] IS NOT NULL;
```

Якщо не у всі поля рядка, що вставляються, вноситься значення, то незаповнені стовпці будуть містити значення NULL.

9.3 Оператор UPDATE

Оператор UPDATEвикористовуєтьсядля оновлення даних у вказаних полях таблиці. Всі записи таблиці, які задовольняють задану у фразі WHERE умову змінюються згідно з фразою SET.

```
UPDATE ім’я_таблиці
SET ім’я_поля=вираз
[,ім’я_поля=вираз]
[WHERE умова];
```

Вираз визначає значення, яке повинно бути встановлене в поле, що оновлюється. При виконанні WHERE будуть змінені лише записи, що задовольняють вказаній умові.

Приклад. Змінити прізвище дівчини “Смаль”, що вийшла заміж, на “Лагус”.

```
UPDATE Студенти
SET Прізвище=”Лагус”
WHERE Прізвище=”Смаль”;
```

Приклад. Ввести до складу клубу всіх студентів із бази *Студенти*, які навчаються з 3-го по 5-ий курс.

```
UPDATE Студенти
SET [Учасник клубу]=True
WHERE [Курс] BETWEEN 3 AND 5;
```

Операція JOINдозволяє об’єднувати дані з кількох таблиць і видавати їх у результуючому наборі даних. Для виконання об’єднання в реченні FROM виконують операцію JOIN. Якщо в результаті вибірки необхідно включити всі рядки з обох таблиць, які задовольняють умові вибірки, використовують операцію INNER JOIN (операція внутрішнього обєднання).

FROM таблиці на запити[{INNER|LEFT|RIGHT} JOIN] ім’я_таблиці ON умова]

Приклад. Вивести дані про всіх студентів, порядкові номери яких збігаються.

```
SELECT *
FROM [Студенти] INNER JOIN [Нові студенти] ON
([Студенти].ID = [Нові студенти].ID))
```

Операції правого та лівого зовнішніх об’єднань вказують на послідовність перерахування таблиць в пропозиції **JOIN.** В деяких випадках це зручно, коли існують головна таблиця і допоміжна, а дані з головної таблиці потрібно отримати в удь – якому випадку.

Приклад. Отримати запит, результатом якого будуть всі записи таблиці *Студенти*. Для тих студентів, які повторюються в таблиці *Нові студенти*, буде додана інформація з цієї таблиці.

```
SELECT *
FROM [Студенти] RIGHT JOIN [Нові студенти] ON
([Студенти].ID = [Нові студенти].ID))
```

Приклад Отримати запит, результатом якого будуть всі записи таблиці *Нові студенти*. Для тих студентів, коди яких повторюються в таблиці *Студенти*, буде додана інформація з цієї таблиці.

```
SELECT *
FROM [Студенти] RIGHT JOIN [Нові студенти] ON
([Студенти].ID = [Нові студенти].ID))
```

Іноді неможливо передбачити, які умови зацікавлять користувача бази даних. Тому є можливість створити запит з неповною умовою.

Приклад. Вивести прізвища всіх студентів, які навчаються на одному курсі, проте номер курсу наперед не заданий, замість цього номера вказується деяка змінна A:

```
SELECT Студенти.Прізвище, Студенти.Місто
FROM Студенти
WHERE (Студенти.Курс = [A])
```

При виконанні запиту, інтерпретатор SQL знаходить всі невизначені змінні та змусить користувача ввести відповідну інформацію

Опис **PARAMETERS** – описує ім’я і тип даних кожного параметра в запиті з параметрами

Приклад.

```
PARAMETERS A Long;
SELECT Студенти.Прізвище, Студенти.Місто
FROM Студенти
WHERE (((Студенти.Курс) = [A]))
```

Приклад Вивести прізвища всіх студентів, які навчаються на одному курсі і мають однакове ім’я.

```
PARAMETERS [Курс] Long, [Ім’я] Text (255);
SELECT Студенти.Прізвище, Студенти.[Ім’я]
FROM Студенти
WHERE (((Студенти.Ім’я])=[Ім’я]) AND ((Студенти.Курс) = [Курс]))
```

Запит з параметрами допомагає автоматизувати процес зміни умов відбору запиту. При наявності запиту з параметрами програма повинна вказувати параметри при кожному запуску запиту. Опис PARAMETERS є необов’язковим, але якщо він

присутній, то повинен знаходитися перед всіма іншими інструкціями, в тому числі перед SELECT.

Інколи необхідно виконувати кілька запитів і об’єднувати результати в одному списку. Оператор UNION додає стрічки списку результатів одного запиту до списку результатів другого, а також виключає значення, що повторюються. Операція дозволяється, коли обидва запити сумісні для з’єднання (містять однакову кількість полів і типи даних відповідних полів співпадають).

Оператор UNION ALL працює як і UNION, але не виключає значень, що повторюються із списку результатів

Питання для контролю

1. Оператор DELETE, особливості застосування.
2. Оператор Insert Into, приклади вставка записів.
3. Оператор Update, оновлення даних у таблиці, особливості.

Лекція 10

ВИЗНАЧЕННЯ ПРАВ ДОСТУПУ КОРИСТУВАЧІВ ДО ДАНИХ

10.1 Користувачі і привілегії

Кожен, хто має доступ до бази даних, називається користувачем. SQL, як правило використовується в багатокористувацьких середовищах, які вимагають розмежування прав користувачів з точки зору доступу до даних і прав на виконання з ними тих чи інших маніпуляцій. З цією метою в SQL реалізовані засоби, що дозволяють встановлювати і контролювати привілегії користувачів бази даних.

Кожен користувач в середовищі SQL має спеціальне ім'я (ідентифікатор), за допомогою якого здійснюється идентифікація користувача з метою встановлення і визначення її прав з точки зору доступу до даних. Кожна надіслана до СКБД команда SQL-запиту асоціюється СКБД з идентифікатором доступу до даних конкретного користувача.

Користувач визначається за допомогою наступної команди:

CREATE USER <ім’я__користувача> IDENTIFIED BY <пароль>.

Після виконання цієї команди користувач стає відомий базі даних, але не може виконувати ніяких операцій.

Видалаляє користувача команда

DROPE USER <ім’я_користувача>.

Призначувані користувачеві привілегії – це дозвіл на виконання зазначеним користувачем даної команди над певним об'єктом бази даних. Існує декілька типів привілеїв, які відповідають кільком типам операцій. Привілеї надаються і скасовуються двома командами SQL відповідно

GRANT – установка привілеїв,

REVOKE – скасування привілеїв.

10.2 Стандартні привілегії

Привілеї, визначені стандартом SQL – це привілеї об'єкта. Це означає, що користувач має привілей (право) на виконання даної команди тільки на певному об'єкті в базі даних. Привілеї об'єкта зв'язані одночасно і з користувачами, і з таблицями бази даних. Тобто, привілегія надається певному користувачеві у вказаній таблиці. Це може бути як базова таблиця, так і представлення.

Користувач, який створив таблицю (будь-якого виду), є власником цієї таблиці. Це означає, що цей користувач має всі привілеї, що відносяться до цієї таблиці, у тому числі він може передавати привілеї на роботу з цією таблицею іншим користувачам.

Користувачеві можуть бути назначені наступні привілеї:

SELECT – користувач може виконувати запити до таблиці.

INSERT – користувач може виконувати в таблиці команду INSERT.

UPDATE – користувач може виконувати в таблиці команду UPDATE. Ця привілегія може бути обмежена визначеними стовпцями таблиці.

DELETE – користувач може виконувати в таблиці команду DELETE.

REFERENCES – користувач може визначити зовнішній ключ (лише для ORACLE), який використовує один чи більше стовпців цієї таблиці в якості батьківського ключа.

Можливе обмеження даної привілегії для визначених стовпців.

Крім того, можуть бути нестандартні привілегії об’єкта, такі як:

INDEX – користувач має право створювати індекс в таблиці.

SYNONYM – користувач має право створювати синонім для об'єкта;

ALTER – користувач має право виконувати команду ALTER TABLE в таблиці.

EXECUTE – дозволяє виконувати процедуру.

Призначення користувачам цих привілеїв здійснюється за допомогою команди GRANT.

10.3 Команда GRANT

Користувач, який є власником, наприклад, таблиці STUDENT може передати іншому користувачеві, наприклад, IVANOV, привілегію SELECT за допомогою команди:

GRANT SELECT ON STUDENT TO IVANOV;

Тепер користувач IVANOV може виконувати SELECT-запити до таблиці STUDENT. Без інших привілегій він може тільки вибирати значення, але не може виконувати будь-які дії, які б впливали на значення в таблиці STUDENT, включаючи використання таблиці STUDENT в якості батьківської таблиці зовнішнього ключа. Коли SQL отримує команду GRANT, перевіряються привілеї користувача, щоб визначити допустимість команди для цього користувача. Користувач IVANOV самостійно не може задати цю команду. Він також не може надати право SELECT іншому користувачеві, так як таблиця йому не належить (нижче буде показано, як власник таблиці може передати іншому користувачеві право надання привілеїв).

Команда

GRANT INSERT ON EXAM_MARKS TO IVANOV;

дозволяє користувачеві IVANOV вводити в таблицю EXAM_MARKS нові рядки.

У команді GRANT допустимо вказувати через коми список привілеїв і список користувачів, яким вони надаються

Наприклад:

GRANT SELECT, INSERTON SUBJECT TO IVANOV, PETROV4

При цьому всі вказані привілеї надаються всім вказаним користувачам. В строгій ANSI-інтерпритації неможливо надати привілеї для кількох таблиць однією командою GRANT.

10.4 Використання аргументів ALL і PUBLIC

Аргумент ALL PRIVILEGES (всі привілеї) або просто ALL використовується замість імен привілеїв в команді GRANT ДЛЯ НАДАННЯ ВСІХ ПРИВІЛЕЇВ В ТАБЛИЦІ.

Наприклад, команда

GRANTALLPRIVILEGESON STUDENT **TO** IVANOV;

або

GRANTALLON STUDENT **TO** IVANOV;

передає користувачеві IVANOV весь набір привілеїв в таблиці STUDENT.

Аргумент **PUBLIC** використовується для передачі вказаних в команді привілеїв всім іншим користувачам. Найбільш часто це застосовується для привілеї SELECT в певних базових таблицях абопредставленнях, які необхідно зробити доступними для будь-якого користувача. Наприклад, щоб дозволити будь-якому користувачеві отримувати інформацію з таблиці EXAM_MARKS, можна застосувати команду

GRANT SELECT ON EXAM_MARKS **TO PUBLIC;**

Надання всіх привілеїв в таблиці всім користувачам, чк правило, є небажаним. Всі привілеї (за виключенням SELECT) дозволяють користувачеві змінювати (чи у випадку **REFERENCES**, обмежувати) вмістиме таблиці, тому дозвіл всім користувачам змінювати вмістиме таблиць може викликати певні проблеми забезпечення безпеки та захисту даних. Тим більше, що привілегія **PUBLIC** не обмежена у передачі прав лише поточним користувачам. Будь-який новий користувач, який додається до системи, автоматично отримує повний набір привілегій, призначений раніше всім користувачам. Тому для обмеження доступу до таблиці всім і завжди краще надати привілегії, відмінні від **SELECT,** лише індивідуальним користувачам.

10.5 Відміна привілегій

Відміна привілегій здійснюється за допомогою команди **REVOKE**, яка має синтаксис, аналогічний команді **GRANT.**

Наприклад, команда

REVOKE INSERT ON STUDENT **FROM** PETROV;

скасовує привілей **INSERT** в таблиці STUDENT для користувача PETROV. Можливе використання в команді **REVOKE** списків привілеїв і користувачів.

Наприклад:

REVOKE INSERT, DELETE ON STUDENT

FROM PETROV, SIDOROV;

Слід мати на увазі, що привілеї скасовуються тим користувачем, який їх надав, при цьому відміна автоматично поширюється на всіх користувачів, що одержали від нього цей привілей.

10.6 Використання представлень для фільтрації привілегій

Дії привілеїв можна зробити більш точними, використовуючипредставлення. При передачі привілегії користувачеві в базовій табляде вона автоматично поширюється на всі рядки, а при використанні можливих вийнятків UPDATE і REFERENCES – і на всі стовпці таблиці. Створюючи представлення, яке посилається на базову таблицю, і потім передаючи привілей вже на цепредставлення, можна обмежити ці привілеї будь-якими виразами у запиті, що міститься упредставленні. Такий метод розширює можливості команди **GRANT.**

Для створення представлень користувач повинен мати привілегію **SELECT**у всіх таблицях, на які він зсилається у представленні. Якщо представлення модифікується, то будь-яка з привілегій INSERT, UPDATE, DELETE, яка представлена користувачу в базовій таблиці, буде автоматично поширюватися напредставлення. Якщо привілеї на оновлення відсутні, то їх неможливо отримати і в створених представленнях, навіть якщо самі ці представлення оновлювані. Так як

зовнішні ключі не застосовуються впредставленнях, то і привілегія **REFERENCES** ніколи не використується при створенні представлень.

Обмеження привілегії SELECT для заданих стовпців

Припустимо, що необхідно забезпечити користувачу PETROV можливість доступу тільки до стовпців STUDENT_ID і SURNAME таблиці STUDENT. Це можна зробити, помістивши імена цих стовпців в представлення

```
CREATE VIEW STUDENT_VIEW AS
    SELECT STUDENT_ID, SURNAME
    FROM STUDENT;
```

і надати користувачу PETROV привілегію **SELECT** у створеному представленні, а не в самій таблиці STUDENT:

```
GRANT SELECT ON STUDENT_VIEW TO PETROV;
```

Для стовпців можна створити різні привілеї, проте слід мати на увазі, що для команди INSERT це буде означати вставку значень за замовчуванням, а для команди DELETE обмеження стовпця не матиме значення.

Обмеження привілегій для заданих стрічок

Представлення дозволяють обмежити (фільтрувати) привілегії для певних рядків таблиці. Для цього використовують в представленні предикат, який визначає включенв в нього рядки. Щоб надати користувачеві PETROV привілегію виду UPDATE в таблиці UNIVERSITY для всіх записів про університети міста Львова, можна створити таке представлення:

```
CREATE VIEW LVIV_UNIVERSITY AS
    SELECT * FROM UNIVERSITY
WHERE CITY = ‘LVIV’
WITH HECK OPTION;
```

Потім можна передати привілегію UPDATE в цій таблиці користувачу PETROV:

GRANT UPDATEON LVIV_UNIVERSITY **TO** PETROV;

На відміну від привілегії **UPDATE** для заданих стовпців, яка поширена на всі рядки таблиці UNIVERSITY, дана привілегія відноситься тільки до рядків, для який значення поля CITY рівне ‘Львів’. Пропозиція **WITHHECKOPTION** захищає користувача PETROV від заміни значення поля CITY на будь-яке значення, крім значення ‘Львів’.

10.7 Інші типи привілегій

Розглянемо питання встановлення ряду інших привілеїв, а саме:

Хто має право створювати таблиці?

Хто має право змінювати, видаляти або обмежувати таблиці?

Чи повинні права на створення базових таблиць відрізнятися від прав на створення представлень?

Чи повинен існувати користувач, який відповідає за підтримку бази даних і, відповідно, має найбільші, або повні привілегії, які не надаються звичайному користувачеві?

Привілеї, які не визначаються в термінах спеціальних об'єктів даних, називаються *привілеями системи* або *правами бази данних*. Ці привілегії включають право створювати об'єкти даних, які відрізняються від базових таблиць (як правило, створюваних кількома користувачами) і представлень (як правило, створюваних більшістю користувачів). Привілеї системи для створення представлень повинні доповнювати, а не заміняти привілегії об'єкта, які вимагає стандарт при створенні представлень. Крім того, в будь-якій системі завжди є користувачі, які мають більшість або всі привілегії і які можуть передати свій статус будь-кому за допомогою привілеї або групи привілеїв. Такого роду користувачем є так званий адміністратор бази даних, або DBA (Data Base Administrator).

10.8 Типові привілегії системи

Загальний підхід визначає три базові привілегії системи:

CONNECT (підключити),

RESOURCE (ресурс)

DBA (адміністратор бази даних).

Привілегія CONNECT складається з права зареєструватися і права створювати представлення і синоніми, якщо передані привілегії об'єкта.

Привілегія RESOURCE складається з права створювати базові таблиці.

Привілегія DBA – це привілегія адміністратора бази даних, тобто користувача, якому надаються найвищі повноваження при роботі з базою даних. Дану привілегію може мати один або більше користувачів з функціями адміністратора бази даних. Команда **GRANT** (у зміненій формі) може застосовуватися як з привілегіями об'єкта, так і з системними привілегіями.

10.9 Створення і видалення користувачів

У більшості реалізацій SQL нового користувача створює користувач з привілегією **DBA**, тобто адміністратор бази даних, який автоматично надає новому користувачу привілегію CONNECT. У цьому випадку, як правило, додається ся пропозицію IDENTIFIED BY, що вказує пароль для цього користувача. Наприклад, команда

GRANT CONNECT TO PETROV **IDENTIFIEDBY** ‘PETROVPASSORD’;

призведе до створення користувача з ім'ям PETROV, надасть йому право реєструватися в базі даних і призначає йому пароль «PETROVPASSORD». Після цього, так як PETROV вже є зареєстрованим користувачем, він (або користувач **DBA**) може використовувати цю ж команду для зміни пароля «PETROVPASSORD».

Коли користувач А надає привілегію **CONNECT** користувачеві В, кажуть, що користувач А «*створює*» користувача В. При цьому користувач А обов'язково повинен мати привілегію **DBA**. Якщо користувач В буде створювати базові таблиці (а не тількипредставлення), йому повинна бути надана і привілегія **RESOURCE**. Але при цьому виникає інша проблема. При спробі видалення користувачем А привілегії **CONNECT** користувача В, який вже має створені ним таблиці, ця команда видалення привілегії буде відхилена, оскільки її дія залишить ці таблиці без власника, що не допускається. Тому, перш ніж видалити привілегію **CONNECT**будь-якому користувачеві, спочатку необхідно видалити з бази даних всі створені цим користувачем таблиці. Привілегію **RESOURCE**видаляти окремо не потрібно, достатньо видалити **CONNECT**, щоб видалити користувача.

Питання для контролю

1. Стандартні привілегії, застосування.
2. Команда Grant, приклади застосування.
3. Як відмінити привілегії?
4. Які є типи привілегій?
5. Команди створення і видалення користувачів

Список літературних джерел.

1. Александр Шпортько, Леся Шпортько. Розробка баз даних в СУБД Microsoft Access / Кондор, 2018.- 184с.

2. Балик Н.Р. Мандзюк В.І. Бази даних MySQL: Навчальний посібник. — Тернопіль: Навчальна книга – Богдан, 2010.— 160 с.

3. Куліків С. Реляційні бази даних у прикладах. Практичний посібник для програмістів і тестувальників. / Кондор, 2019. – 175с.

4. Гайдаржи В.І., Ізварін І.В. Бази даних в інформаційних системах / В-во: Університет "Україна", 2018 – 418с.

5. Булатецька Л В. Мова запитів SQL : текст лекцій нормативної навчальної дисципліни "Бази даних та розподілені інформаційно-аналітичні системи" / Булатецька Леся Віталіївна, Булатецький Віталій Вікторович. – Луцьк : СНУ імені Лесі Українки, 2018. – 92

6. Рамський Ю.С., Цибко Г.Ю.Проектування й опрацювання баз даних: Посібник для вчителів. / Богдан, 2010.-116 с.

7. https://www.embarcadero.com/products/delphi

Зміст