ДЕРЖАВНИЙ УНИВЕРСИТЕТ ТЕЛЕКОМУНІКАЦІЙ

Кафедра системного аналізу

ЯРЦЕВ В.П.

Організація баз даних та знань

Навчальній посібник

Київ - 2018

Рецензенти:

Циганок В.В., доктор технічних наук, с.н.с., завідувач лабораторії систем підтримки прийняття рішень Інституту проблем реєстрації інформації НАН України.

Гумен О.М, доктор технічних наук, професор, Національний технічній університет України «Київський політехнічний інститут ім. Ігоря Сікорського».

Ярцев В.П.
Організація баз даних та знань: навчальний посібник.-К. ДУТ 2018.-214с.

В навчальному посібнику розглядається систематизована сукупність відомостей що до понять баз даних та інформаційних систем, основні етапи та принципи побудови реляційних баз даних. Викладено математичні основи розробки об'єктів баз даних, оператори мови структурованих запитів SQL, Pl/SQL, методика розробки інформаціоно – логічної моделі бази засобами CASE-технологій AllFusion ERwin Data Modeler 7.2, фізичне проектування у IBExpert2.5. Розглянуті особливості архітектурі «клієнт-сервер», питання розробки клієнтської та серверної частин інформаційної системи на прикладі проектування реляційних, об'єктно – орієнтованих баз даних та баз знань на прикладі СУБД Microsoft Access2016, серверів БД InterBase 2017, MySQL, IBM DB2 Express-C9.7. Приведено основи побудови об'єктно – орієнтованих програм візуального моделювання додатків до баз даних у середовищі програмування Embarcadero Delphi XE5, Visual Studio C#.

Посібник призначений для студентів, що навчаються за усіма спеціальностями напрямку галузі - Інформаційні технології, Телекомунікація і радіотехніка.

Обговорено та схвалено на засіданні кафедри системного аналізу
Протокол №__9___ від __12.04. 2018р.

Рекомендовано методичною радою факультету Інформаційних технологій ДУТ до друку
Протокол №_____ від _____2018р.

Зміст

Короткий термінологічний словник

Адаптер — пристрій, що дає змогу з'єднувати фізично неоднорідні системи (наприклад, плата, яка забезпечує зв'язок шини комп'ютера *з* зовнішнім пристроєм.

Адміністратор бази даних — людина, яка відповідає за розміщення, збереження, цілісність, використання інформації БД, мережі та надає відповідні повноваження користувачам.

Алгоритм — послідовність дій, що визначає процес перетворення інформації від початкових даних до кінцевого результату.

База даних — пойменована сукупність даних, які мають однакові принципи опису, збереження, оброблення інформації.

Байт — основна одиниця кількості інформації, містить 8 біт.

Біт — мінімальна одиниця кількості інформації (один символ двійкового алфавіту).

Буфер обміну — область пам’яті для тимчасового збереження інформації.

База знань – сукупність знань, що відносяться до деякої предметної області, формально представлених так, щоб на їх основі можна було здійснювати міркування.

Введення-виведення — пересилання даних між оперативною пам'яттю і зовнішніми пристроями.

Вікно бази даних Access — вікно, що з'являється на екрані дисплея та містить перелік її об'єктів (таблиць, запитів, форм, звітів, макросів, модулів).

Вузол — пристрій в мережі (ним можуть бути робоча станція, принтер або файловий сервер).

Гіперпосилання — текст, що містить посилання, теля активізації яких здійснюється перехід до іншого файлу (текстового, графічного).

Глобальна мережа (WAN) — мережа, елементи якої розташуються на значній віддалі.

Дозвіл — установлення обмежень для користувачів, які мають доступ до деякого об'єкта системи, і режим цього доступу (повний, обмежений, недозволений).

Домен — сукупність робочих станцій і серверів мережі, що адмініструється як єдина група (кожен домен має певні межі безпеки).

Драйвер — програма, що встановлює додаткові параметри.

Запис — група взаємозв'язаних елементів, що розглядається як єдине ціле.

Захист даних — апаратні та програмні засоби для запобігання втратам або порушенню цілісності даних.

Інтерфейс — сукупність засобів і правил, які забезпечують логічну або фізичну взаємодію пристроїв та програм обчислювальної системи.

Ключ (ключове поле) — поле, значення якого ідентифікує запис (у ключовому полі кожен елемент даних повинен мати унікальне значення).

Код — засіб перетворення інформації з однієї системи на іншу (наприклад, символьної на двійкову систему числення).

Командний файл — файл, що містить послідовність команд мовою програмування і виконується у пакетному режимі.

Комірка — найменша структурна одиниця, що використовується для збереження даних .

Транзакція - група логічно зв'язаних послідовно виконуваних операцій.

Експертна система – це комплекс комп'ютерного програмного забезпечення, що допомагає людині приймати обґрунтовані рішення.

Список скорочень

ADO - (Microsoft Active Data Objects) компоненти для доступу к БД
BDE - (Borland Database Engine) процесор БД фірми Borland
DLL - бібліотека динамічно завантажуваемих процедур та функцій
DDL - (Data definition language) - мова опису даних
DWH - (DataWarehouse) – сховище даних
CASE - (Computer-Aided Software Engineering) система автоматизованого проектування
ODBC - (Open Data Base Connectivity) відчинений інтерфейс підключення до баз даних
SQL - (Structured Queries Language) структурована мова запитів
БД - база даних
ІС - інформаційна система
КБД - корпоративна БД
НФ - нормальна форма
НД - набір даних
НФ - нормальна форма
ООП - об'єктне орієнтоване програмування
ОС - операційна система
ПЕОМ - персональна електронна обчислювальна машина
ПБД - персональна БД
СУБД - система управління базами даних

Вступ

Інформаційні системи (ІС) та бази даних сталі обов'язковою частиною у всіх галузях суспільства: в економіці, науці, культурі, промисловості та інших. Зараз взагалі важко знайти область діяльності людини, у якій у тім чи іншому виді не застосовувалися б бази даних (БД). Сучасні тенденції в області побудови і розвитку корпоративних інформаційних систем спрямовані в бік інтеграції існуючих автоматизованих систем, і на основі єдиної інтеграційної шини будуються композитні додатки що до роботи з ІС. Очевидним фактом є те, що більшість сучасних ІС розробляються на базі інтернет-технологій з використанням сучасних технологічних платформ і стандартів взаємодії. Постійно зростаючі потреби інформатизації суспільства з одного боку, і вибуховий процес розвитку комп'ютерних технологій, з іншого боку, обумовлюють і швидкі темпи розвитку теорії і практики побудови БД. Тому при вивченні дисципліни " Організація баз даних та знань " важливо головну увагу приділяти принципам побудови БД і сутності технологій, що вже одержали широке поширення у сучасної практиці. З обліком цього у дисципліну включені наступні 10 тем.

Тема 1. Основні поняття баз даних. Реляційна модель даних.

Основні визначення і поняття. Поняття БД і СУБД. Типи архітектури БД: локальні (автономні), файл-серверні, клієнт-серверні і багатоярусні БД. Реляційна модель даних. Поняття відносин, сутності, атрибутів. Таблиця БД – форма представлення відносин. Первинний ключ. Поняття індексу. Індексування таблиць БД. Первинний і вторинний індекси. Зв'язування таблиць. Види зв'язків "один до одного", "один до багатьох" і "багато хто до багатьох". Ключ зв'язку. Поняття посилальної цілісності даних. Проектування БД. Нормалізація відносин. 1, 2 і 3 нормальні форми.

Тема 2. Розробка концептуальної моделі баз даних CASE засобами.

Метод "сутність-зв'язок" при проектуванні БД. Створення інформаційно логічних моделі БД. Архітектура розподілених баз даних. Проектування розподілених БД. Розподілення даних. Фрагментація. Реплікація. Прозорість розподілення. Система автоматизованого проектування бізнесу-процесу AllFusion Process Modeler 7.2: інтерфейс, панелі інструментів, ціль моделювання, побудова моделі IDEF0, діаграми дерева вузлів і FEO, каркас діаграми, злиття й розщеплення моделей, створення звітів в BPwin. Створення інформаційно логічної моделі проекту реляційної БД за допомогою CASE-технологій AllFusion ERwin Data Modeler 7.3.

Тема 3. Мова SQL. Архітектура "клієнт-сервер"

Елементи мови SQL: оператори створення об'єктів БД - DDL, оператори модифікації даних DML. Поняття запиту. Види запитів. Оператор SELECT: загальний синтаксис; вибірка записів з однієї таблиці. Умови добору; вибірка записів з декількох таблиць. Внутрішнє з'єднання таблиць; групування даних. Групові (агрегатні) функції і їхнє застосування в запитах. Архітектура "клієнт-сервер". Проблеми доступу до даних багатьох користувачів. Поняття транзакції. Керування транзакціями. Блокування, рівні ізоляції транзакцій.

Сервер БД InterBase 7: склад, технічні характеристики, Типи даних, масиви. Створення БД і її псевдоніма. Створення таблиць БД. Первинні ключі та генератори. Індекси, обмеження БД, представлення. Процедури, тригери. Створення уявлень та процедур БД. Утиліти IBConsol. Моделювання та аналіз БД, створювання бази даних з допомогою програми CASE-технологій IBExpert. Робота з типом даних BLOB

Сервер БД MySQL: склад, технічні характеристики, Типи даних. Створення файлу та таблиць, первинних ключів та зовнішнього зв'язку. Створення уявлень та процедур, індексів, обмежень БД . Утиліта phpMyAdmin. Захист БД.

Тема 4. Адміністрування та побудова баз даних у сервері IBM DB2 Express

Загальні відомості, призначення і можливості СУБД IBM DB2 Express. Встановлення, запуск серверу. Типи даних. Створення таблиць БД. Імена полів. Властивості полів. Індекси, обмеження БД, представлення. Поняття запиту. Види запитів. Оператор SELECT: загальний синтаксис; вибірка записів з однієї таблиці. Умови добору; вибірка записів з декількох таблиць. Внутрішнє з'єднання таблиць; групування даних. Групові (агрегатні) функції і їхнє застосування в запитах. Поняття транзакції. Керування транзакціями. Процедури, тригери. Створення уявлень та процедур БД. Утиліти DB2. Створення первинного ключа; зв'язування

таблиць БД (створення схеми даних); Робота з даними. Уведення даних. Уведення даних у таблиці БД; пошук і сортування даних; фільтрація даних. Запити. Схема даних і бланк запиту; застосування в запиті групових операцій. запити з параметрами; запуск, збереження, модифікація запитів. Оператори керування та захисту БД.

Тема 5. Створення додатків до БД у системі об'єктно – орієнтованого програмування DelphiXE, VS C#.

Поняття класу й об'єкта. Ієрархія базових класів Delphi; поняття компонента. Бібліотека візуальних компонентів. Властивості і події компонентів; поняття проекту Delphi. Склад проекту; створення простого проекту. Компіляція проекту і запуск на виконання. Створення в середовищі Delphi додатків БД. BDE – основа додатка БД Delphi. Поняття псевдоніма БД. Загальна схема взаємодії додатка з БД; створення псевдоніма БД за допомогою утиліти BDEADMIN; основні компоненти Delphi для роботи з БД; основні етапи створення додатка БД - розміщення і настроювання компонентів 1) набору даних; 2) джерела даних; 3) компонента відображення даних. Набори даних. Програмування роботи з даними: поняття набору даних (НД). Компоненти TTable, TQuery. Стану НД. Навігація по НД; клас TField. Програмний доступ до полів; використання Редактора полів. Редагування існуючих і створення нових полів: що обчислюються і полів підстановки; компонент TQuery; зв'язування НД компонентів TTable і Tquery.

Тема 6. Загальні принципи побудови розподілених баз даних

Архітектура розподілених баз даних. Топології РСУБД. Розподілена обробка даних. Паралельні СУБД. Системи з поділами пам'яті. Системи з поділами дисків. Системи без поділів. Гомогенні та гетерогенні розподілені. СУБД. Мультибазові системи. Етапи проектування розподілених БД. Розподілення даних. Фрагментація даних. Локальність посилань, надійність і доступність даних, рівень продуктивності. Реплікація даних. Функції служби реплікації. Прозорість розподілення. Схеми володіння даними. Збереження цілісності транзакцій. Тригерні процедури. Виявлення і вирішення конфліктів транзакцій.

Тема 7. Структура, адміністрування та створювання баз даних у сервері Oracle DB 11g

Загальні відомості, склад, можливості, архітектура сервера БД Oracle 11g. Інсталяція Oracle. Табличний простір, схема БД. Глобальна системна область. Фонові процеси. Основні об'єкти сервера. Процедури користувача. Типи даних Oracle11g. Обробка виключених ситуацій. Зв'язок з вилученою БД. Проектування РБД у СКБД Oracle11 g. Створення структур таблиць. Додання даних у таблиці. Опис програм. Створення та робота з пакетами. Створення тригерів БД. Підключення та робота в комп'ютерної мережи. Управління користувачами. Безпека БД.

Тема 8. Організація баз знань

Основні поняття і визначення баз знань. Моделі та концепції проектування баз знань. Витягання знань, глибинне знання, експертна система, екстенсионал поняття, знання, керована ціль, керований антецедент, керований консеквент, машина виведення, поверхневе знання, придбання знань, формування знань, інтенсионал поняття. Логіко - лінгвістічна модель, мережа, моделювання, предикат, продукційна модель, пропозиціональна функція, семантика, терм, умова, фрейм. Проектування інформаційно – логічної моделі баз знань.

Тема 9. Створення та робота з об'єктно-орієнтованою БД у сервері Cache2015

Архітектура серверу БД Caché2015, процедуру створення і налаштування БД і областей Caché. Основні поняття об'єктно-орієнтованої технології (клас, об'єкт, види атрибутів класу, принципи поліморфізму, спадкоємства і інкапсуляції). Основні типи класів Caché і сфера їх застосування, типи властивостей Caché і принципи роботи з ними. Необхідні команди і функції мови програмування Caché Object Script. Типи методів Caché і принципи їх написання і використання. Різні типи запитів Caché SQL (динамічний, статичний, вбудований) і уміти ефективно настроювати індекси.

Тема 10. Створення WEB додатків клієнтів баз даних засобами Inter Systems Cache

Архітектура технології Caché Server Pages і принципи реалізації веб – додатків. Конфігурування Caché Server Pages Web Gateway. Етапи створення додатку щодо БД. Робота з змінними CSP. Використання форм. Керування сеансом. Становлення зв'язку з БД. Захист інформації користувача. Оновлення інформації у БД.

1. Основні поняття та визначення баз даних

1.1 Основні визначення та класифікація інформаційних систем

Сучасне суспільство неможливе представити без широкого застосування різних інформаційних систем. Інформація в сучасному світі перетворилася в один з найбільш важливих ресурсів, а інформаційні системи (ІС) стали необхідним інструментом практично у всіх сферах діяльності. Наведемо наступні визначення.

Під **інформаційною системою** звичайно розуміють сукупність технічних і програмних засобів, призначених для рішення задач збору, обробки і представлення інформації користувачу в необхідному виді.

Наприклад, це задачі обліку і руху матеріальних засобів, системи електронних платежів, карткова система сплати телефонних дзвінків, інформаційно-довідкові задачі та інше. В кожному з наведених прикладів завжди обов'язково присутні два елемента ІС: база даних, у якої повинна зберігатися інформація, яка потрібна для вирішення конкретних задач, і засоби управління цією базою даних.

База даних (БД) – це сукупність спеціальним образом організованих даних, що зберігаються у пам'яті обчислювальної системи, і об'єктів, що відображають стан деякої предметної області.

Система управління базами даних (СУБД) – це комплекс мовних і програмних засобів, призначених для створення, ведення і спільного використання БД багатьма користувачами.

Знання - це виявлені закономірності наочної області (принципи, зв'язки, закони), що дозволяють вирішувати завдання в цій області. При обробці на ЕОМ знання трансформуються аналогічно даним:

- знання в пам'яті людини як результат мислення;
- матеріальні носії знань (підручники, методичні посібники);
- поле знань - умовний опис основних об'єктів наочної області, їх атрибутів і закономірностей, що їх зв'язують;
- знання, описані на мовах подання знань (продукційні мови, семантичні мережі, фрейми);
- бази знань.

База знань – сукупність знань, що відносяться до деякої предметної області, формально представлених так, щоб на їх основі можна було здійснювати міркування. Бази знань найчастіше використовуються в контексті експертних систем, де з їх допомогою подаються навики і досвід експертів, зайнятих практичною діяльністю у відповідної області (наприклад, в медицині або в математиці). Зазвичай база знань є сукупністю правил виводу.

Розмаїтість задач, розв'язуваних за допомогою ІС, привело до появи безлічі різнотипних систем, що відрізняються принципами побудови і закладеними в них правилами обробки інформації. Організації зіткнулися із вибуховим зростанням інформації щодо клієнтів, довкілля і взаємодії людей, предметів і систем (рис.1.1).

Рисунок 1.1 - Динаміка зростання та роль використання інформації у суспільстві

Інформаційні системи можна класифікувати по цілому ряді різних ознак (рис.1.2). В основу розглянутої класифікації покладені найбільш істотні ознаки, що визначають функціональні можливості та особливості побудови сучасних систем. У залежності від обсягу розв'язуваних задач, використовуваних технічних засобів, організації функціонування, інформаційні системи поділяються на ряд груп.

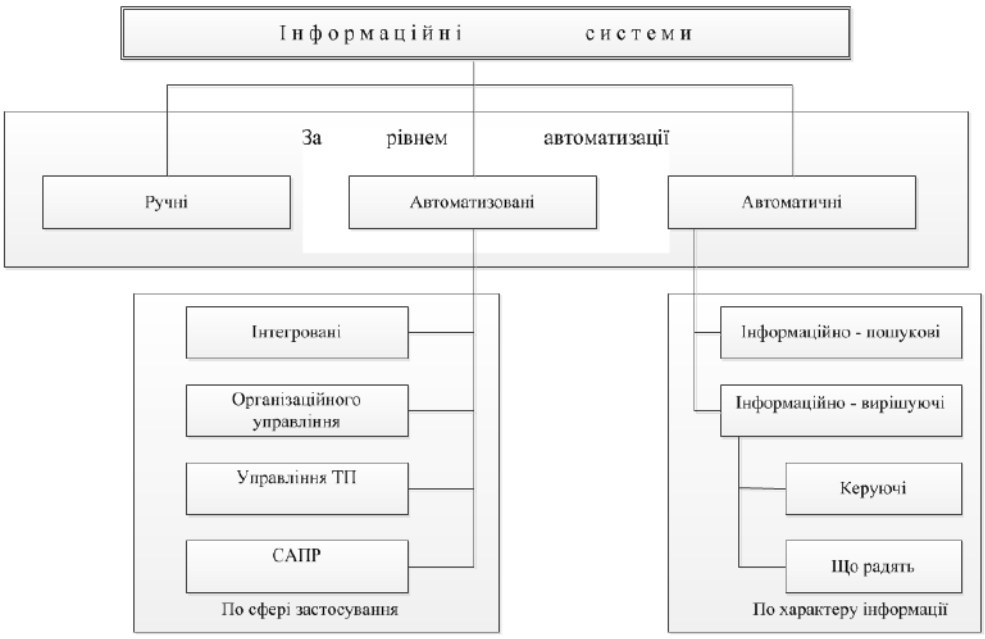


Рисунок 1.2 - Класифікація інформаційних систем

По типі збережених даних ІС поділяються на фактографічні і документальні. Фактографічні системи призначені для збереження та обробки структурованих даних у виді чисел і текстів. Над такими даними можна виконувати різні операції. У документальних системах інформація представлена у виді документів, що складаються з найменувань, описів, рефератів і текстів. Пошук по неструктурованим даним здійснюється з використанням семантичних ознак. Відібрані документи надаються користувачеві, а обробка даних у таких системах практично не виробляється. Ґрунтуючись на ступені автоматизації інформаційних процесів у системі керування підприємством, інформаційні системи поділяються на ручні, автоматичні і автоматизовані. Ручні ІС характеризуються відсутністю сучасних технічних засобів переробки інформації і виконанням всіх операцій людиною. В автоматичних ІС всі операції по переробці інформації виконуються без участі людини.

Автоматизовані ІС припускають участь у процесі обробки інформації і людини, і технічних засобів, причому головна роль у виконанні рутинних операцій обробки даних приділяється комп'ютерові. Саме цей клас систем відповідає сучасному поняття "інформаційна система".

У залежності від характеру обробки даних ІС поділяються на інформаційно-пошукові й інформаційно-вирішальні. Інформаційно-пошукові системи роблять уведення, систематизацію, збереження, видачу інформації з запиту користувача без складних перетворень даних. (Наприклад, ІС бібліотечного обслуговування, резервування і продажі квитків на транспорті, бронювання місць у готелях і ін.) Інформаційно-вирішальні системи здійснюють, крім того, операції переробки інформації з визначеного алгоритму.

По характері використання вихідної інформації такі системи прийнята поділяти на керуючі і що радять. Результуюча інформація керуючих ІС безпосередньо трансформується в прийняті людиною рішення. Для цих систем характерні задачі розрахункового характеру й обробка великих обсягів даних. (Наприклад, ІС планування виробництва або замовлень, бухгалтерського обліку.) ІС що радять виробляють інформацію, що приймається людиною до зведення і врахо-

вується при формуванні управлінських рішень, а не ініціює конкретні дії. Ці системи імітують інтелектуальні процеси обробки знань, а не даних. (Наприклад, експертні системи.)

У залежності від сфери застосування розрізняють наступні класи ІС. Інформаційні системи організаційного керування - призначені для автоматизації функцій управлінського персоналу як промислових підприємств, так і непромислових об'єктів (готелів, банків, магазинів і ін.). Основними функціями подібних систем є: оперативний контроль і регулювання, оперативний облік і аналіз, перспективне й оперативне планування, бухгалтерський облік, керування збутому, постачанням і інші економічні й організаційні задачі. ІС керування технологічними процесами (ТП) - служать для автоматизації функцій виробничого персоналу по контролі і керуванню виробничими операціями. У таких системах звичайно передбачається наявність розвитих засобів виміру параметрів технологічних процесів (температури, тиску, хімічного складу і інше), процедур контролю допустимості значень параметрів і регулювання технологічних процесів.

ІС автоматизованого проектування (САПР) - призначені для автоматизації функцій інженерів-проектувальників, конструкторів, архітекторів, дизайнерів при створенні нової техніки або технології. Основними функціями подібних систем є: інженерні розрахунки, створення графічної документації (креслень, схем, планів), створення проектної документації, моделювання проектованих об'єктів.

Інтегровані (корпоративні) ІС - використовуються для автоматизації усіх функцій фірми і охоплюють весь цикл робіт від планування діяльності до збуту продукції. Вони містять ряд модулів (підсистем), що працюють у єдиному інформаційному просторі і виконуючих функціях підтримки відповідних напрямків діяльності.

Аналіз сучасного стану ринку ІС показує стійку тенденцію росту попиту на ІС організаційного керування. Причому попит продовжує рости саме на інтегровані системи керування. Автоматизація окремої функції, наприклад, бухгалтерського обліку або обліку збуту готової продукції, вважається вже пройденим етапом для багатьох підприємств.

Існує класифікація ІС у залежності від рівня керування, на якому система використовується. ІС оперативного рівня - підтримує виконавців, обробляючи дані про угоди і події (рахунка, накладні, зарплата, кредити, потік сировини і матеріалів). ІС оперативного рівня є сполучною ланкою між фірмою і зовнішнім середовищем. Задачі, мети, джерела інформації й алгоритми обробки на оперативному рівні заздалегідь визначені й у високому ступені структуровані.

ІС фахівців - підтримують роботу з даними і знаннями, підвищують продуктивність і продуктивність роботи інженерів і проектувальників. Задача подібних інформаційних систем - інтеграція нових зведень в організацію і допомогу в обробці паперових документів.

ІС рівня менеджменту - використовуються працівниками середньої управлінської ланки для моніторингу, контролю, прийняття рішень і адміністрування. Основні функції цих інформаційних систем:

- порівняння поточних показників з минулими;
- складання періодичних звітів за визначений час, а не видача звітів по поточних подіях, як на оперативному рівні;
- забезпечення доступу до архівної інформації і інше.

Стратегічна ІС - комп'ютерна інформаційна система, що забезпечує підтримку прийняття рішень по реалізації стратегічних перспективних цілей розвитку організації. Інформаційні системи стратегічного рівня допомагають вищій ланці керівників вирішувати неструктуровані задачі, здійснювати довгострокове планування. Основна задача - порівняння змін, що відбуваються в зовнішнім оточенні, з існуючим потенціалом фірми. Вони покликані створити загальне середовище комп'ютерної телекомунікаційної підтримки рішень у зненацька виникаючих ситуаціях. Використовуючи самі зроблені програми, ці системи здатні в будь-який момент надати інформацію з багатьох джерел. Деякі стратегічні системи володіють обмеженими аналітичними можливостями.

З погляду програмно-апаратної реалізації можна виділити ряд типових архітектур ІС. Традиційні архітектурні рішення засновані на використанні виділених файлів-серверів або серверів баз даних. Існують також варіанти архітектурі корпоративних ІС, що базуються на технології Internet (Intranet-додатки). Наступний різновид архітектури інформаційної системи ґрунтується на концепції "сховища даних" (DataWarehouse) - інтегрованого інформаційного середо-

вища, що включає різнорідні інформаційні ресурси. Для побудови глобальних розподілених інформаційних додатків використовується архітектура інтеграції інформаційно-обчислювальних компонентів на основі об’єктне - орієнтованого підходу.

1.2 Різновиди архітектурі БД

Розрізняють такі основні типи організації (архітектури) БД: автономні (локальні), файл-серверні, клієнт-серверні, багаторівневі, розподілена, інтернет- орієнтована (рис.1.3). Розглянемо коротко особливості кожної з цих архітектур.

Рисунок 1.3 - Архітектури сучасних баз даних

Автономні (локальні) БД є найбільш простим типом організації БД.
Характерними рисами автономних БД є наступне:
- БД і СУБД розміщені на одному ПК (рис. 1.4);
- мережа не використовується;
- з даними в кожний момент часу працює тільки один користувач.

Рисунок 1.4 - Автономна (локальна) БД

Автономні БД широко застосовуються на невеликих підприємствах для бухгалтерського і кадрового обліку, а також і окремими користувачами для збереження й обробки власних даних. В останньому випадку говорять іноді про персональні БД (ПБД). Однак, у сучасному житті найбільш розповсюджені ІС, у яких доступ до даних здійснюється через комп'ютерну мережу. У таких ІС до однієї БД можуть одночасно звертатися багато користувачів. Історично першими з'явилися ІС із БД, що використовують архітектуру «файл-сервер». Нагадаємо тут зміст понять *сервер* і *клієнт*, що далі часто використовуються.

Взагалі *сервером* деякого ресурсу прийнято називати комп'ютер (чи програму), який (яка) керує цим ресурсом. Ресурсом може бути не тільки пам'ять, процесорний час ПК, але також і, наприклад, файлова система, служба печатки, БД і інше. Якщо ресурсом є БД, то програма, що керує цим ресурсом, називається *сервером БД*.

Клієнтом звичайно називають комп'ютер (програму), що використовує відповідний ресурс. Іноді *клієнтом* називають також людину-користувача, яка працює з програмою-клієнтом.

Для архітектури «файл-сервер» характерним є те, що дані зберігаються на окремому комп'ютері, який називають файловим сервером.

Найчастіше це мережний сервер. Структура ІС, заснованої на архітектурі «файл-сервер», зображена на рис. 1.5.

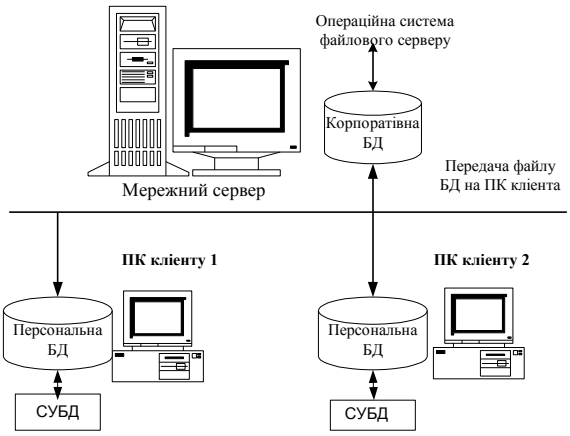


Рисунок 1.5 - Архітектура «файл-сервер».

У відповідь на запити користувачів-клієнтів із сервера на їх ПК через мережу пересилаються файли з даними. Обробка даних виконується на ПК клієнтів.

Недоліки такої архітектури наступні:

- низька продуктивність системи, обумовлена тим, що клієнту пересилаються надлишкові дані. Наприклад, клієнту можуть знадобитися дані тільки з 1 запису з таблиці розміром 100000 записів. Проте при даній архітектурі клієнту на його ПК пересилається весь файл із 100000 записів. Клієнт (його програма) потім сам відшукує потрібний йому запис на своєму ПК. При збільшенні числа клієнтів швидко настає перевантаження мережі і продуктивність системи різко знижується;
- велика імовірність внесення помилок у дані, що можуть порушити цілісність даних і негативно вплинути на роботу інших клієнтів. Причина цього в тім, що з даними безпосередньо працюють програми численних клієнтів, що можуть «по-своєму» виконувати дії з даними, нарешті, можуть вносити “свої” помилки;
- низька захищеність даних, яка обумовлена тим, що БД у цьому випадку являє собою просто набір файлів (файл) на мережному серверу, доступ до яких регулюється тільки мережною операційною системою. Це робить БД по суті беззахисною від випадкових чи навмисних перекручувань, від знищення чи розкрадання інформації.

Ці і інші суттєві недоліки архітектури «файловий-сервер» привели до необхідності розробки більш досконалої архітектури, яку було названо архітектурою «клієнт-сервер».

В архітектурі «клієнт-сервер» БД так само розміщується на мережному серверу, але основна обробка даних виконується не на ПК клієнта, а на цьому ж мережному серверу за допомогою спеціального програмного забезпечення (ПЗ) сервера БД (рис. 1.6).

Функціювання інформаційної системи, яка застосовує архітектуру «клієнт-сервер», відбувається наступним чином.

При одержанні від клієнта запиту (мовою *SQL*) сервер БД виконує пошук потрібних даних, чи виконує якісь дії над даними і повертає клієнту тільки необхідну йому інформацію (а не великі файли з даними, як в архітектурі «файл-сервер»).

При цьому істотно підвищується продуктивність системи за рахунок того, що знижується завантаження каналів зв'язку, а обробка даних виробляється значно швидше, тому що комп'ютер сервера, як правило, є більш могутнім.

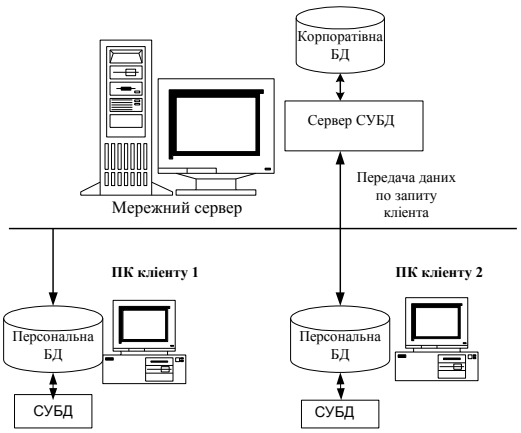


Рисунок 1.6 - Архітектура «клієнт-сервер»

Крім того, ПЗ сервера БД є оптимізованим для швидкого виконання операцій по пошуку інформації і забезпечує надійний контроль за цілісністю даних. Крім того, на серверу БД також передбачені спеціальні засоби для захисту інформації БД від несанкціонованого доступу, від навмисного чи ненавмисного пошкодження даних.

Багаторівнева архітектура БД є подальшим розвитком архітектури «клієнт-сервер» стосовно до великих розподілених БД. Найбільше поширення одержала трьохрівнева архітектура, суть якої полягає в наступному:

- На нижньому рівні на ПК клієнтів установлюється програма клієнта, що забезпечує інтерфейс з користувачем. Ніякої обробки даних на цьому рівні не виконується. Користувачу-клієнту представляється тільки результуюча інформація, отримана у відповідь на його запит;
- На другому рівні розташований сервер прикладних програм, який забезпечує обмін даними між користувачем і видаленою БД. Сервер прикладних програм розміщається у вузлі мережі, доступному всім клієнтам;
- На третьому рівні розташований видалений сервер БД, на якому зберігаються БД. Він приймає інформацію (запити) від серверів прикладних програм і керує ними.

Достоїнства трьохрівневої архітектури полягають у наступному:

- зниження навантаження на сервер;
- суттєве спрощення клієнтських програм;
- єдине поводження клієнтів, що зменшує вірогідність помилок;
- можливість роботи клієнтів на різних платформах.

Це найбільш складна і гнучка схема організації БД, що одержує усе більше поширення у великих розподілених ІС.

1.3 Схема обміну даними у ПК при роботі з БД

Будь-якому користувачу, що працює з БД, дуже важливо мати представлення про обчислювальний процес, що відбувається в ОС при роботі з БД. Розглянемо найбільш важливі складові внутрішнього механізму процесів роботи з БД на декількох прикладах. Якщо користувач взаємодіє із СУБД через спеціально розроблений додаток то обмін інформацією (рис.1.7). Цикл взаємодії користувача з БД через додаток БД можна розділити на наступні основні етапи. Користувач у процесі діалогу з додатком БД уводить з термінала запит на необхідні дані.

Додаток БД на програмному рівні засобами мови маніпулювання даними (SQL) формує запит до БД. СУБД перевіряє права (паролі) користувача. Потім відповідно до форматів даної СУБД перетворить інформацію запиту до форматів команд файлової системи ОС. СУБД засобами доступу файлової системи ОС робить пошук інформації в БД, зчитує шукані дані і поміщає їх у системні буфери СУБД. Отримані дані СУБД перетворить до необхідного формату і пересилає їх у відповідні області пам'яті програми-додатка БД.

Додаток БД отримані дані (результати) відображає в необхідному виді на терміналі користувача. Достоїнством схеми є –простота розробки і ведення БД, додатків за допомогою убудованих коштів.

Недолік – необхідність витрат дискової пам'яті для збереження програми СУБД і додатка.

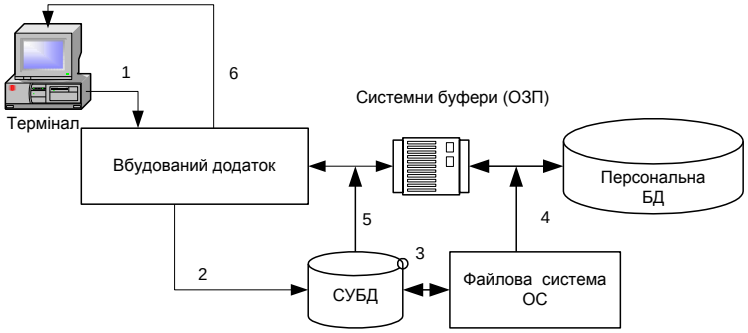


Рисунок 1.7 - Схема доступу до БД із додатком БД

У випадку, якщо СУБД містить у собі і функції інтерфейсу користувача, то користувач звертається до БД коштами тільки СУБД, не використовуючи для цього спеціальні додатки БД. Наприклад, за такою схемою відбувається робота з автономною БД у СУБД Access. При цьому схема обміну даними має вид, показаний на рис.1.8. Ця схема несуттєво відрізняється від розглянутої вище. Основні принципи реалізації процесів обміну даними залишаються такими ж. Функції взаємодії з користувачем забезпечуються самої СУБД.

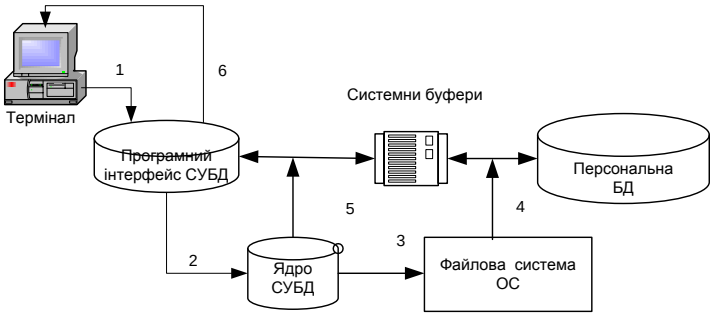


Рисунок 1.8 - Схема доступу до БД коштами СУБД

У даному випадку досягається зменшення розміру HDD і ОЗУ, необхідне для збереження програми СУБД, підвищується швидкість роботи додатка, забезпечується можливість модифікації додатка для зручності користувача.

У випадку потреби забезпечення захисту додатка від змін з боку користувача використовують схему з додатком і ядром СУБД (рис.1.9).

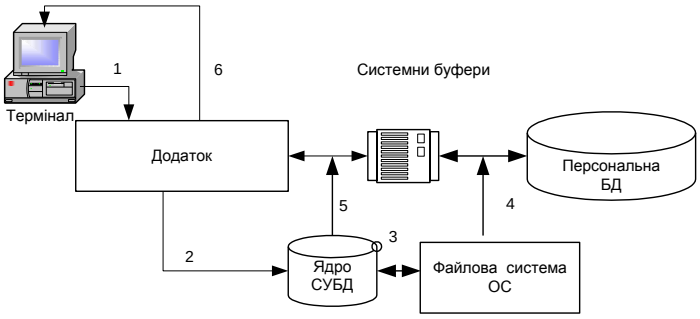


Рисунок 1.9 - Схема доступу до БД засобами додатка і ядра СУБД

Так MS Access включає додатковий пакет Developer's Toolkit, за допомогою якого можна створити укорочену (утримуюче тільки ядро) версію Access, не утримуючих інструментів розробки додатків.

1.4 Тенденції розвитку сучасних СУБД

У світі в даний час розроблені і застосовуються донині сотні різних СУБД. Усі СУБД можна класифікувати по виду використовуваної програми , характеру і моделі даних. Найбільш розповсюдженими з них є наступні.

Полнофункциональні СУБД, що підтримують локальні і загальні БД з архітектурою «файл-сервер», що мають інтерфейс, що дозволяє створювати, модифікувати структури таблиць, уводити дані, формувати запити, розробляти звіти, виводити їх на печатку: ***Visual FoxPro, dBASE, Paradox, Access*** і ін.

Богатокористувальніцкі багатофункціональні СУБД, що включають у себе як можливості сервера БД, так і клієнтів: ***Oracle, Informix, SyBase*** і ін.

Сервери БД призначені для організації центрів обробки даних в архітектурі «клієнт - сервер». Обмін із клієнтськими програмами здійснюється за допомогою операторів SQL. Прикладами є програми: ***MS SQL Server (MicroSoft), InterBase (Embarcadero), MySQL (Oracle), IBM DB2, Oracle Database, Cache (Inter Systems).***

Клієнтськими програмами для серверів БД можуть бути повнофункціональні СУБД (***Access, FoxPro***), електронні таблиці (***Excel***), текстові процесори (***Word***), програми електронної пошти. Взаємодія користувача із СУБД відбувається через його чи інтерфейс за допомогою спеціально розробленого додатка, що дозволяє вводити дані, формувати запити до БД і образжати результати пошуку інформації.

Додатки можуть бути убудовані в СУБД і незалежні (розроблені за допомогою інших програм). В існуючих СУБД при розробки додатків використовують:

- ручне кодування програм (***Access, IBM DB2, MS SQL Server Oracle, Cache***);
- створення текстів додатків за допомогою генераторів (***Application Express Oracle)***.
- автоматична генерація готового додатка за допомогою програм візуального програмування(форми ***Access, Oracle, Cache***).

Створені будь-яким методом додатки, не можуть виконуватися без СУБД, що аналізує уміст файлів додатка та автоматично створює машинні команди. Цей процес зветься - інтерпретація, використовується в ***Access, Oracle***.

Найбільш розповсюдженими засобами для розробки додатків БД у даний час є: ***Delphi, C#, C++ Builder, Visual Basic, Java .***

Основні направлення та тенденції розвитку сучасного програмного забезпечення для проектування та супроводження БД наступні:

- пошук сучасних моделей зберігання інформації, впровадження нових типів даних в базах;
- розробка нових архітектур СУБД , що забезпечує можливість зберігати і обробляти дані щодо обсягів до петабайт;
- розширення областей застосування БД: опрацювання надвеликих обсягів інформації; розподіленої обробки інформації в мережі.
- забезпеченні інформаційного обслуговування мобільного користувача.

Контрольні питання.

1. Визначення інформаційної системи і її можливості.
2. Призначення СУБД.
3. Основні архітектури побудови БД.
4. Основні схеми обміну даними, їхнього достоїнства і недоліки.
5. Що називається сервером БД?
6. Що називається клієнтом БД?
7. За допомогою яких програм створюються додатки до БД?
8. Які СУБД використовують у архітектурі «клієнт – сервер»?

2. Реляційні бази даних

2.1 Моделі зберігання даних

Збережені в БД записі мають логічну структуру – модель представлення даних. Найбільш розповсюдженими моделями раніше були ієрархічна, мережна, реляційна. В даний час розроблені нові моделі: постреляційна, багатомірна, об'єктно-орієнтована. На їхній основі створюються комбіновані моделі: об’єктно -реляційна, дедуктивне - об’єктно - орієнтовані, семантичні, концептуальні. У деяких СУБД підтримується одночасно кілька моделей даних (***Cache***).

Перш, ніж перейти до вивчення реляційних систем БД, необхідно ознайомиться з до реляційними СУБД так як внутрішня організація реляційних систем багато в чому заснована на використанні методів ранніх систем, це буде корисно для поняття шляхів розвитку постреляціних СУБД. Обмежуємося розглядом тільки загальних підходів до організації трьох типів ранніх систем:

- систем, заснованих на інвертованих списках;
- ієрархічних ;
- мережних систем керування базами даних;
- Об’єктно орієнтовані базами даних.

Почнемо з деяких найбільш загальних характеристик ранніх систем. Ці системи активно використовувалися протягом багатьох літ, довше, ніж використовується яка-небудь з реляційних СУБД. Насправді деякі з ранніх систем використовуються навіть у наш час, накопичені величезні бази даних, і однієї з актуальних проблем інформаційних систем є використання цих систем разом із сучасними системами. Усі ранні системи не ґрунтувалися на яких-небудь абстрактних моделях.

Поняття моделі даних фактично узвичаїлося фахівців в області БД тільки разом з реляційним підходом. Абстрактні представлення ранніх систем з'явилися пізніше на основі аналізу і виявлення загальних ознак у різних конкретних систем. У ранніх системах доступ до БД вироблявся на рівні записів. Користувачі цих систем здійснювали явну навігацію в БД, використовуючи мови програмування, розширені функціями СУБД. Інтерактивний доступ до БД підтримувався тільки шляхом створення відповідних прикладних програм із власним інтерфейсом. Навігаційна природа ранніх систем і доступ до даних на рівні записів змушували користувача самого робити всю оптимізацію доступу до БД, без якої-небудь підтримки системи. Після появи реляційних систем більшість ранніх систем булии оснащені "реляційними" інтерфейсами. Однак у більшості випадків це не зробило їх по-справжньому реляційними системами, оскільки залишалася можливість маніпулювати даними в природному для них режимі.

2.1.1 БД які засновані на інвертованих списках

До числа найбільш відомих і типових представників таких систем відносяться ***Datacom/DB*** компанії ***Applied Data Research, Inc. (ADR)***, орієнтована на використання на машинах основного класу фірми IBM, і ***Adabas*** компанії ***Software AG***.

Організація доступу до даних на основі інвертованих списків використовується практично у всіх сучасних реляційних СУБД, але в цих системах користувачі не мають безпосереднього доступу до інвертованих списків (індексів). База даних, організована за допомогою інвертованих списків, схожа на реляційну БД, але з тією відмінністю, що збережені таблиці і шляхи доступу до них видні користувачам. Рядки таблиць упорядковані системою в деякій фізичній послідовності. Фізична упорядкованість рядків усіх таблиць може визначатися і для всієї БД (так робиться, наприклад, у Datacom/DB). Для кожної таблиці можна визначити довільне число ключів пошуку, для яких будуються індекси. Ці індекси автоматично підтримуються системою, але явно видні користувачам.

Підтримуються два класи операторів. Першій клас це оператори, що встановлюють адресу запису, серед яких різняться прямі пошукові оператори (наприклад, знайти перший запис таблиці по деякому шляху доступу) та оператори, що знаходять запис у термінах відносної по-

зиції від попереднього запису по деякому шляху доступу. Другий клас складають оператори над записами яки адресуються.

Типовий набір операторів має вигляд:

- LOCATE FIRS T - знайти перший запис таблиці T у фізичному порядку (повертає адреса запису);
- LOCATE FIRS T WITH SEARCH KEY EQUAL - знайти перший запис таблиці T із заданим значенням ключа пошуку K (повертає адреса запису);
- LOCATE NEXT - знайти перший запис, що випливає за записом із заданою адресою в заданому шляху доступу (повертає адреса запису);
- LOCATE NEXT WITH SEARCH KEY EQUAL - знайти наступний запис таблиці T у порядку шляху пошуку з заданим значенням K, повинне бути відповідність між використовуваним способом сканування і ключем K (повертає адресу запису);
- LOCATE FIRST WITH SEARCH KEY GREATER - знайти перший запис таблиці T у порядку ключа пошуку K зі значенням ключового поля, великим заданого значення K (повертає адреса запису);
- RETRIVE - вибрати запис із зазначеним адресом;
- UPDATE - обновити запис із зазначеним адресом;
- DELETE - видалити запис із зазначеним адресом;
- STORE - уключити запис у зазначену таблицю; операція генерує адресу запису.

Загальні правила визначення цілісності БД відсутні. У деяких системах підтримуються обмеження унікальності значень деяких полів, але в основному усіх покладається на прикладну програму.

2.1.2 Ієрархічні структури БД

Найбільш відомої і розповсюдженим є ***Information Management System (IMS)*** фірми IBM. Перша версія з'явилася в 1968 р. Дотепер підтримується багато баз даних, що створює істотні проблеми з переходом як на нову технологію БД, так і на нову техніку.

Ієрархічна БД складається з упорядкованого набору декількох екземплярів одного типу графа (дерева). Для опису структури ієрархічної БД на деякій мові програмування використовується тип даних «дерево».

Тип дерева схожий з типом даних « запис » алгоритмічної мови програмування Паскаль. Він складається з одного "кореневого" типу запису та упорядкованого набору з нуля або більших типів піддерева (кожне з яких є деяким типом дерева). Тип дерева в цілому являє собою ієрархічно організований набір типів запису.

Приклад типу дерева (схеми ієрархічної БД) наведений на рис. 2.1

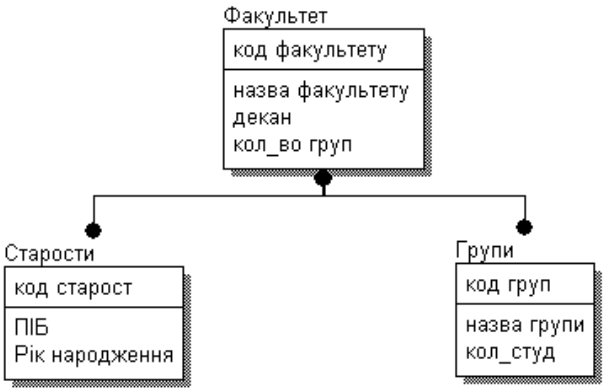


Рисунок 2.1 - Приклад схеми ієрархічної БД

Тут таблиця ***Факультет*** є кореневим типом (предком) для таблиць ***Старости*** і ***Групи***, а ***Старости*** і ***Групи*** – підлеглі типи (нащадки) таблиці ***Факультет***.

Між типами запису підтримуються зв'язки. Всі екземпляри даного типу нащадку з загальним екземпляром типу предку називаються близнюками. Для БД визначений повний порядок обходу - униз, праворуч. У IMS використовувалася оригінальна і нестандартна термінологія: "сегмент" замість "запис", а під "записом БД" розумілося все дерево сегментів.

Прикладами типових операторів маніпулювання ієрархічно організованими даними можуть бути наступні:

- Знайти зазначене дерево БД;
- Перейти від одного дерева до іншого;
- Перейти від одного запису до інший усередині дерева;
- Перейти від одного запису до іншої в порядку обходу ієрархії;
- Уставити новий запис у зазначену позицію;
- Видалити поточний запис.

Автоматично підтримується цілісність посилань між предками і нащадками.

Основне правило: ніякий нащадок не може існувати без свого батька.

Аналогічна підтримка цілісності по посиланнях між записами, що не входять в одну ієрархію, не підтримується. В ієрархічних системах підтримувалася деяка форма представлень БД на основі обмеження ієрархії.

До достоїнств даної моделі даних відносяться ефективне використання пам'яті ЕОМ і високі показники часу виконання основних операцій над даними.

Недоліками ієрархічної моделі є її громіздкість для обробки інформації з досить складними логічними зв'язками і складність у розумінні для користувача.

2.1.3 Мережні системи БД

Типовим представником є ***Integrated Database Management System (IDMS***) компанії ***Cullinet Software.Inc***., призначена для використання на машинах основного класу фірми IBM під керуванням більшості операційних систем. Архітектура системи заснована на пропозиціях ***Data Base Task Group (DBTG)*** Комітету з мов програмування ***Conference on Data Systems Languages (CODASYL)***, організації, відповідальної за визначення мови програмування Кобол.

Мережний підхід до організації даних є розширенням ієрархічного. В ієрархічних структурах запис - нащадок повинна мати в точності одного предка; у мережній структурі даних нащадок може мати будь-як число предків.

Мережна БД складається з набору записів і набору зв'язків між цими записами, а якщо говорити більш точно, з набору екземплярів кожного типу з заданого в схемі БД набору типів запису і набору екземплярів кожного типу з заданого набору типів зв'язку. Простий приклад мережної схеми БД наведена на рис.2.2.

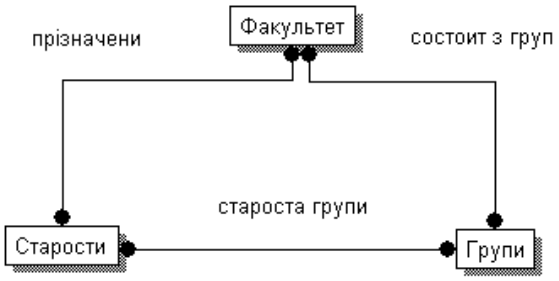


Рисунок 2.2 - Приклад мережної схеми БД

Тип зв'язку визначається для двох типів запису: предка і нащадка. Екземпляр типу зв'язку складається з одного екземпляра типу запису предка й упорядкованого набору екземплярів типу

запису нащадка. Для даного типу зв'язку L з типом запису предка P і типом запису нащадка C повинні виконуватися наступні дві умови:

- Кожен екземпляр типу P є предком тільки в одному екземплярі L;
- Кожен екземпляр C є нащадком не більш, ніж в одному екземплярі L.

На формування типів зв'язки не накладаються особливі обмеження; можливі, наприклад, що випливають ситуації. Тип запису нащадка в одному типі зв'язку L1 може бути типом запису предка в іншому типі зв'язку L2 (як в ієрархії):

- даний тип запису P може бути типом запису предка в будь-якому числі типів зв'язку;
- даний тип запису P може бути типом запису нащадка в будь-якому числі типів зв'язку;
- може існувати будь-як число типів зв'язку з тим самим типом запису предка і тим самим типом запису нащадка;
- якщо L1 і L2 - два типи зв'язку з тим самим типом запису предка P і тим самим типом запису нащадка C, те правила, по яких утвориться споріднення, у різних зв'язках можуть розрізнятися;
- типи запису X і Y можуть бути предком і нащадком в одному зв'язку і нащадком і предком - в іншій;
- предок і нащадок можуть бути одного типу запису.

Зразковий набір операцій може бути наступної:

- Знайти конкретний запис у наборі однотипних записів (студента Сидорова);
- Перейти від предка до першого нащадка по деякому зв'язку (до першого студенту групи);
- Перейти до наступного нащадка в деякому зв'язку (від Сидорова до Іванова);
- Перейти від нащадка до предка по деякому зв'язку (знайти групу де навчається Сидоров);
- Створити новий запис;
- Знищити запис;
- Модифікувати запис;
- Включити в зв'язок;
- Виключити зі зв'язку;
- Переставити в інший зв'язок і інше.

Підтримка обмеження цілісності в принципі не потрібно, але іноді вимагають цілісності по посиланнях (як в ієрархічній моделі).

До достоїнствам раннього СУБД можна віднести:

- Розвиті засоби керування даними в зовнішній пам'яті на низькому рівні;
- Можливість побудови вручну ефективних прикладних систем;
- Можливість економії пам'яті за рахунок поділу подоб’єктів (у мережних системах).

Основними недоліки є:

- Складність у використанні, необхідні знання про фізичну організацію збереження даних;
- Прикладні системи залежать від цієї організації і їхнього логіка перевантажена деталями організації доступу до БД.

2.1.4 Об’єктно-орієнтована модель БД

Об’єктно-орієнтована розробка програмного забезпечення базується на ключовому принципі об’єктної технології – моделюванні застосувань у вигляді об’єктів програмного забезпечення, що описують як властивості так і поведінку об’єктів реального світу. Об’єкти такого роду інкапсульовані та скривають всю складність внутрішньої структури даних і реалізації поведінки за загальнодоступними інтерфейсами. Завдяки цьому, об’єкти можуть використовуватися в різних програмних пакетах без необхідності вивчення їх внутрішньої реалізації. Взаємозв’язок об’єктно-орієнтованих застосувань з БД висуває особливі вимоги до СКБД і середовища програмування, особливо якщо необхідно уникнути характерних втрат продуктивності та ускладнення семантики.

Загальновизнані реляційні СКБД орієнтовані на збереження простих даних, в той час як відображення об’єктів складної структури пов’язано з великими витратами. Намагання викори-

стати технологію реляційних БД в таких складних системах, як автоматизоване проектування (computer aided design, CAD), автоматизоване виробництво (computer aided manufacturing, CAM), технологія програмування, системи, основані на знаннях, мультимедійні системи висвітлили обмеженість систем реляційних БД.

В умовах, коли з'явилося нове покоління застосувань БД, виникла необхідність забезпечити збереження об'єктів, яка б найкращим чином задовольнялася при використанні об'єктно-орієнтованих (постреляційних) баз даних ООБД (ПРБД).

2.2 Реляційна модель даних

Дані, що зберігаються в БД, завжди мають визначену логічну структуру, тобто описується тією чи іншою моделлю даних. Ми будемо вивчати тільки основні поняття реляційної моделі даних, тому що саме вона одержала широке поширення і застосовується практично у всіх промислових СУБД. В основі реляційної моделі лежить поняття ***відношення***, саме тому модель називається реляційною (від англійського ***relation*** – відношення).

Відношенням R називається підмножина декартова добутку множин $D_1, D_2,...,D_n$.

Математично це записується так:

$R \subseteq D_1 \times D_2 \times ... \times D_n$,

де R – ім'я відносини;

$D_1 \times D_2 \times ... \times D_n$ - повний декартовий добуток множин, на яких побудоване відношення R.

Множини $D_1, D_2, ..., D_n$ називаються **доменами**.

Повний декартовий добуток – це набір усіляких сполучень з n елементів, де кожен елемент береться зі свого домену.

Розглянемо простий приклад. Нехай задані множини-домени:

D_1 = {Іванов, Петров};

D_2 = {Фізика, Математика};

D_3 = {3, 4, 5}.

Тоді повний декартовий добуток цих множин-доменів наступний:

$D_1 \times D_2 \times D_3$ = {⟨Іванов, Фізика, 3⟩,
⟨Іванов, Фізика, 4⟩,
⟨Іванов, Фізика, 5⟩,
⟨Іванов, Математика, 3⟩,
⟨Іванов, Математика, 4⟩,
⟨Іванов, Математика, 5⟩,
⟨Петров, Фізика, 3⟩,
⟨Петров, Фізика, 4⟩,
⟨Петров, Фізика, 5⟩,
⟨Петров, Математика, 3⟩,
⟨Петров, Математика, 4⟩,
⟨Петров, Математика, 5⟩}.

Відзначимо зараз, що кутові дужки ⟨⟩ у математиці прийнято застосовувати для представлення упорядкованих наборів даних, якими є кортежі, а фігурні дужки {} - для представлення множин, які по визначенню неупорядковані.

Число елементів повного декартового добутку у наведеному прикладі дорівнює $n_1 \times n_2 \times n_3 = 2 \times 2 \times 3 = 12$, де n_1, n_2, n_3 – кількості елементів у доменах D_1, D_2, D_3.

Відношення R завжди є представленням деякої реальної ситуації і тому може містити тільки деякі з можливих комбінацій.

Наприклад, відношення може містити тільки три елементи:

R = {⟨Іванов, Фізика, 5⟩, ⟨Іванов, Математика, 4⟩, ⟨Петров, Фізика, 3⟩}.

У даному відношенні представлена ситуація, коли Іванов має по однієї оцінці з фізиці і математиці, а Петров має оцінку тільки по фізиці (математику він ще не здавав).

Число вихідних множин-доменів називають **ступенем відносини**. Якщо число вихідних множин = 2, відношення називається **бінарним.**

Якщо число вихідних множин *n*, то ***n*-арним.**

Сутністю називається об'єкт будь-якої природи, дані про який зберігаються в БД. Дані про сутність у БД зберігаються у виді відносини.

Атрибутами називають властивості, що характеризують сутність.

У розглянутому вище прикладі відносини сутність, яка у ньому представлена, можна назвати "Успішність студентів". Атрибутів для цієї сутності ми ввели три: **Прізвище, Предмет, Оцінка**. Домени D_1, D_2, D_3 – це множини можливих значень відповідних атрибутів.

Кортежом називається упорядкована сукупність значень атрибутів, що характеризують окремий екземпляр (об'єкт) даної сутності.

У приведеному вище прикладі: ⟨Іванов, Фізика, 5⟩ - це кортеж, що описує конкретний екземпляр сутності "Успішність студентів" - успішність студента Петрова.

Відношення має просту графічну інтерпретацію – воно може бути представлене у виді таблиці. Наприклад:

Успішність студентів		
Прізвище	Предмет	Оцінка
Іванов	Фізика	5
Іванов	Математика	4
Петров	Фізика	3

Стовпці в таблиці відповідають різним атрибутам і їх (стовпці) називають часто *входженнями доменів* у дане відношення. **Рядки** таблиці є окремими *кортежами* відносини.

У таблицях БД стовпці часто називають **полями**, а рядки – **записами.**

Заголовок таблиці часто називають **схемою відношення**. Наприклад, схема відношення "Успішність студентів" має наступний вигляд:

Успішність студентів		
Прізвище	Предмет	Оцінка

Таблиці, у яких представляються відношення, називають **реляційними таблицями**. Однак, не всяка таблиця є реляційною. Реляційні таблиці повинні задовольняти наступним вимогам:

1. У таблицях не може бути однакових рядків (кортежів).
2. Заголовки стовпців (імена атрибутів) повинні бути унікальними.
3. Усі значення в одному стовпці повинні бути однорідними, тобто мати один тип даних.
4. Порядок розміщення рядків у таблиці довільний.

Для забезпечення унікальності кортежів у відношенні визначається первинний ключ.

Первинним ключем відносини називають атрибут (чи сукупність атрибутів), що однозначно ідентифікує (визначає) кожен кортеж. Значення первинного повинні бути *унікальними*, тобто не можуть повторюватися в різних кортежах. Завдяки наявності первинного ключа у відношенні ніколи не можуть з'явитися однакові кортежі (однакові рядки у таблиці).

Між відношеннями (таблицями) у реляційних БД практично завжди встановлюються зв'язки, які відображують реальні зв'язки між сутностями. Встановлення зв'язків між відношеннями необхідно *для забезпечення цілісності даних*.

Зв'язок між відношеннями встановлюється через атрибути, які називають **ключами зв'язку** (чи **зовнішніми ключами**). Значення ключів в зв'язаних кортежах повинні бути однаковими. Ключі (первинні і зовнішні) можуть бути **простими** (з одного атрибута) чи **складеними** (складатися з декількох атрибутів). Загальна класифікація ключів представлена на рис.2.3.

Рисунок 2.3 - Класифікація ключів реляційної моделі БД

Таким чином, реляційну БД можна розглядати як множину взаємозв'язаних відношень. Основні переваги реляційної моделі даних:

1. Таблична форма організації даних, що є великою зручністю як для розроблювачів, так і для користувачів БД.
2. Строге теоретичне обґрунтування моделі, засноване на математичній теорії відношень.

Причому, останнє, очевидно, є найбільш важливим. Зупинимося коротко на деяких основних поняттях математичної теорії відношень.

2.3 Операції над відношеннями. Реляційна алгебра

В основі математичної теорії відношень лежить реляційна алгебра. Нагадаємо, що взагалі **алгеброю** називається множина визначених об'єктів із заданої на ньому сукупністю операцій, замкнутих щодо цієї множини об'єктів. Цю множину називають **основною множиною** алгебри. Замкнутість операцій означає, що результат кожної з операцій завжди є елементом тієї ж основної множини, з якої вибираються операнди.

У реляційній **алгебрі** (її також називають реляційною алгеброю Кодда по імені вченого, що запропонував її) основною множиною є **множина відношень.**

Множина операцій реляційної алгебри включає наступні 8 операцій: об'єднання, перетинання, різниця, добуток, вибірка, проекція, ділення, з'єднання.

Операндами реляційних операцій є відношення. Операції можуть бути **бінарними** (виконуватися над двома відношеннями) чи **унарними** (виконуватися над одним відношенням).

Для деяких бінарних операцій обов'язковим є вимога **сумісності структур відношень**, до яких застосовується дана операція. Відношення є сумісними за структурою, якщо вони мають однаковий ступінь і атрибути відношення можуть бути впорядковані таким чином, що на однакових місцях у схемі відношень будуть розташовуватися порівнянні атрибути, тобто атрибути, що приймають значення з тих же самих доменів.

Далі розглянемо стисло суть кожної з операцій реляційної алгебри.

Об'єднанням $R_1 \cup R_2$ двох відношень R_1 і R_2 називається відношення, яке є множиною кортежів, які одночасно присутні в R_1 і R_2 і не повторюються.

Приклад:

R₁		
Прізвище	**Предмет**	**Оцінка**
Іванов	Математика	5
Петров	Фізика	3
R₂		
Прізвище	**Предмет**	**Оцінка**
Іванов	Математика	5
Іванов	Фізика	4

$R_1 \cup R_2$		
Прізвище	**Предмет**	**Оцінка**
Іванов	Математика	5
Петров	Фізика	3
Іванов	Фізика	4

Перетинанням $R_1 \cap R_2$ двох відношень R_1 і R_2 називається відношення, що містить тільки ті кортежі, що належать одночасно першому і другому відношенням.

Приклад: для тих же відношень R_1 і R_2 перетинанням є відношення:

$R_1 \cap R_2$		
Прізвище	**Предмет**	**Оцінка**
Іванов	Математика	5

Різницею $R_1 \backslash R_2$ двох відношень R_1 і R_2 називається відношення, що містить множину кортежів таких, що належать R_1, але не приналежних R_2. Приклад:

$R_1 \backslash R_2$		
Прізвище	**Предмет**	**Оцінка**
Петров	Фізика	3

Операції об'єднання і перетинання є комутативними, тобто їхній результат не залежить від порядку проходження операндів: $R_1 \cup R_2 = R_2 \cup R_1$, $R_1 \cap R_2 = R_2 \cap R_1$.

Операція різниці є не комутативна, тобто її результат залежить від порядку проходження операндів: $R_1 \backslash R_2 \neq R_2 \backslash R_1$.

Добутком $R_1 \otimes R_2$ називається таке відношення, усі кортежі якого отримані шляхом *зчеплення* кожного кортежу з R_1 з кожним кортежем з R_2. При цьому ступінь результуючого відношення дорівнює $n+m$, де n – ступінь відношення R_1, а m - ступінь відношення R_2. Операція добутку може бути застосовна до відношень, не обов'язково сумісних за структурою.

Приклад:

R_1	
Група	**Прізвище**
АД-21	Іванов
АД-22	Петров

R_2	
Предмет	**Оцінка**
Математика	5
Фізика	4

$R_1 \otimes R_2$			
Група	**Прізвище**	**Предмет**	**Оцінка**
Д-21	Іванов	Математика	
Д-22	Петров	Математика	
Д-21	Іванов	Фізика	
Д-22	Петров	Фізика	

У даному прикладі у результуючому відношенні кортеж ⟨АД-21, Іванов, Математика, 5⟩ є зчепленням кортежів: ⟨АД-21, Іванов⟩ і ⟨Математика, 5⟩.

Вибіркою з відношення R за умовою f називається нове відношення, що складається з кортежів, узятих з R і задовольняючих умові відбору f. Операція вибірки записується наступним чином: R WHERE f. Умова відбору f являє собою логічний вираз, у якому операндами є імена атрибутів чи константи, а операціями – логічні операції: **not** (“не”), **or** (“чи”), **and** (“і”), чи операції порівняння: $=, \neq, <, >, \geq, \leq$. Логічний вираз f може приймати одне з двох значень: *Істина* (*True*) – умова виконується, чи *Неправда* (*False*) – умова не виконується. Розглянемо приклад:

R		
Прізвище	**Адреса**	**РікНар**
Іванов	Київ	1982

Петров	Бровари	1980
Козлов	Київ	1981

R WHERE **РікНар = 1981**		
Прізвище	**Адреса**	**РікНар**
Козлов	Київ	1981

Проекцією *R[X,Y,Z,...]* відношення *R* на атрибути *X,Y,Z,…*, де множина *{X,Y,Z,...}* є підмножиною повної множини атрибутів *R*, називається нове відношення з атрибутами *X,Y,Z,…*, яке містить частини кортежів, узятих з *R*. Кортежі, що повторюються, з результуючого відношення виключаються. Приклад:

R			
Прізвище	Предмет	Оцінка	Дата
Іванов	Фізика	5	5.10.02
Іванов	Математика	3	3.11.02
Петров	Математика	4	8.10.02
Петров	Філософія	4	15.11.02

Проекція відношення *R* на атрибути **Прізвище** й **Оцінка** буде наступною:

R **[Прізвище, Оцінка]**	
Прізвище	**Оцінка**
Іванов	5
Іванов	3
Петров	4

Операція ділення. Нехай є двоє відношень: R_1 з атрибутами **A** і **B**, і R_2 з атрибутом **B**, причому атрибут **B** є загальним для обох відношень, і визначений на тому ж самому домені. Атрибути **A** і **B** взагалі можуть бути складеними. Результатом ділення R_1 на R_2 буде нове відношення *R* таке, що містить атрибут **A** і множину кортежів ⟨a⟩ таких, що у відношенні R_1 маються кортежі ⟨a, b⟩, причому множина значень {b} у цих кортежах повинна включати всі значення {b} з R_2. Суть операції продемонструємо на наступному прикладі:

R_1	
A	**B**
a1	b1
a1	b2
a1	b3
a1	b4
a1	b5
a2	b1
a2	b2
a3	b2
a4	b2
a4	b4

R_2
B
b2
b4

$R=R_1/R_2$
A
a1
a4

В результуюче відношення *R* включено кортеж ⟨a1⟩ тому, що вихідне відношення R_1 містить множину кортежів {⟨a1, b2⟩, ⟨a1, b4⟩}, у якої множина значень атрибута **B** {b2, b4} включає всі значення **B**, які маються в R_2. Кортеж ⟨a2⟩ не включено в результуюче відношення тому, що у відношенні R_1 відсутня множина {⟨a2, b2⟩, ⟨a2, b4⟩}, а є тільки один кортеж ⟨a2, b2⟩.

Операція з'єднання має кілька різновидів. Загальна схема операції з'єднання відношень R_1 і R_2 за умовою *f* наступна. На початку будується декартовий добуток множин R_1 і R_2.

Потім до отриманого результату застосовується операція вибірки за заданою умовою *f*. Для пояснення сутності операції з'єднання розглянемо такий простий приклад.

Нехай задані наступні відношення R_1 і R_2:

R_1	
Прізвище	**Адреса**
Іванов	Київ
Петров	Бровари

R_2		
Прізвище	**Предмет**	**Оцінка**
Іванов	Фізика	5
Іванов	Математика	3
Петров	Фізика	4
Петров	Математика	4

Потрібно одержати прізвища й адреси студентів, що мають оцінки “4”. Дана задача може бути вирішена шляхом виконання операції з'єднання відношень R_1 і R_2 за умовою:

f = (R_1.**Прізвище** = R_2.**Прізвище**) **and** (R_2.**Оцінка** = 4).

Операція реалізується в такий спосіб.

На початку формується декартовий добуток:

$R_1 \otimes R_2$				
Прізвище	**Адреса**	**Прізвище**	**Предмет**	**Оцінка**
Іванов	Київ	Іванов	Фізика	5
Іванов	Київ	Іванов	Математика	3
Іванов	Київ	Петров	Фізика	4
Іванов	Київ	Петров	Математика	4
Петров	Бровари	Іванов	Фізика	5
Петров	Бровари	Іванов	Математика	3
Петров	Бровари	Петров	Фізика	4
Петров	Бровари	Петров	Математика	4

Потім до отриманого відношення застосовується операція вибірки за умовою f. Після цього одержимо:

$R = R_1 \otimes R_2$ WHERE f			
Прізвище	**Адреса**	**Предмет**	**Оцінка**
Петров	Бровари	Фізика	4
Петров	Бровари	Математика	4

У результуюче відношення включається тільки один атрибут **Прізвище**, тому що він є загальним для відношень, що з'єднуються. По заданій умові потрібно одержати тільки прізвища й адреси. Тому до останнього відношення (назвемо його R) застосуємо операцію проекція на атрибути **Прізвище** й **Адреса**:

R [Прізвище, Адреса]	
Прізвище	**Адреса**
Петров	Бровари

Кортежі, що повторюються, з результуючого відношення виключаються.

Контрольні запитання:

1. Поняття відношення, кортеж, домен. Надати визначення, навести приклади.
2. Поняття реляційної таблиці.
3. Первинні та зовнішні ключі відношення.
4. Що таке реляційна алгебра? Особливості операцій реляційної алгебри.
5. Операції об’єднання, перетинання і різниці відношень. Навести приклади.

3. Проектування баз даних

3.1. Проблеми і етапи проектування баз даних

Процес проектування БД – це тривалий багато етапний процес, у результаті якого буде побудована працююча ІС, за допомогою якої будуть вирішуватися саме ті задачі, для яких вона розроблялася. У загальному випадку можна виділити наступні етапи процесу проектування:

Системний аналіз предметної області і словесний опис інформаційних об'єктів і зв'язків між ними.

Інфологічне проектування – побудова інформаційно-логічної (інфологічної) моделі предметної області.

Вибір СУБД.

Даталогічне проектування – логічне проектування, засноване на даних з урахуванням специфікацій обраної СУБД.

Фізичне проектування – вибір ефективного способу розміщення БД на зовнішніх носіях для забезпечення найбільш ефективної роботи прикладних програм.

Далі розглянемо коротко сутність кожного з цих етапів.

3.1.1 Системний аналіз предметної області

Системний аналіз предметної області повинний дозволити скласти докладний словесний опис основних об'єктів даної предметної області, їхніх властивостей і зв'язків між цими об'єктами. При цьому розрізняють такі два підходи до системного аналізу: функціональний і структурний.

Функціональний підхід заснований *на аналізі задач*, що повинні бути вирішуватися користувачем чи організацією, для якої створюється інформаційна система. При такому підході визначаються об'єкти і зв'язки між ними, найбільш важливі й істотні для забезпечення рішення тільки поставлених задач. У цьому випадку велика імовірність, що виділені об'єкти і зв'язки будуть недостатні для рішення інших задач, що можуть з'явитися в майбутньому при розвитку організації.

Структурний підхід не фіксує жорстко інформаційні потреби користувача. В опис предметної області включаються усі найбільш характерні і важливі для неї об'єкти і взаємозв'язки. У цьому випадку БД виявляється придатної для рішення у майбутньому множини задач, у тому числі і таких, котрі раніше і не ставилися. Такий підхід представляється дуже привабливим, однак, прагнення до повного охоплення предметної області може привести до надлишкове складної схеми БД, що може привести до зниження ефективності вирішення окремих конкретних задач.

На практиці рекомендується використовувати компромісний варіант, який, з одного боку, орієнтований на конкретні задачі користувачів, а, з іншого боку, враховує прогнозовані потреби нарощування можливостей прикладних програм, які можуть з'явитися у майбутньому.

Системний аналіз повинний закінчуватися докладним описом *інформації про об'єкти* предметної області, що потрібні для рішення конкретних задач, коротким описом *алгоритмів рішення цих задач*. При цьому також повинні бути описані *вихідні документи*, що повинні генеруватися в системі, і опис *вхідних документів*, що є джерелом даних, що вводяться в БД.

3.1.2 Інфологічне проектування

Метою інфологічного (інформаційно-логічного) проектування є побудова інформаційно-логічної моделі предметної області. Ця модель повинна в напівформалізованому вигляді представляти схему проекту майбутньої БД. При цьому ступінь деталізації її повинна бути достатньої для наступного її "перекладу на мову" конкретної реляційної СУБД, і, одночасно, вона повинна легко читатися неспеціалістом, яким завжди є замовник – майбутній користувач БД. Це

завжди дуже важливо, тому що на цьому етапі можуть вирішуватися питання про фінансування створення БД.

В даний час найбільш розповсюдженим є підхід до інформаційно-логічного проектування, заснований на застосуванні моделі “сутність-зв'язок” (*ER*-моделі). Уже розроблені і застосовуються на практиці інструментальні засоби, які використовують модель “сутність-зв'язок” (*CASE*-системи), за допомогою яких автоматизується процес розробки інформаційно-логічної моделі. Нижче ми більш докладно познайомимося з основними положеннями моделі “сутність-зв'язок”.

3.1.3 Вибір СУБД

Рішення про вибір СУБД залежить від багатьох факторів, аналіз яких виходить за рамки нашого курсу. Найбільш важливими факторами тут є масштаб створюваної інформаційної системи і вимоги до її швидкодії. Важливе значення також має наявний в організації досвід застосування СУБД.

3.1.4 Даталогічне проектування

Даталогічне проектування (логічне проектування, засноване на даних) виконується на етапі, коли уже визначена інформація, яка повинна бути представлена в БД, і вже обрана конкретна СУБД, тобто уже відомі припустимі типи даних, обмеження і синтаксичні правила опису відносин.

Якщо раніше вже була побудована інфологічна модель, то на даному етапі вона перетворюється в схему відношень реляційної БД. Виконується аналіз коректності схеми відношень. Аналіз цей заснований на застосуванні методу нормальних форм, який дозволяє виявити й усунути небажані залежності між атрибутами у відношеннях, що може привести у майбутньому до низької ефективності робота з БД. Суть цього методу буде розглянута нижче, в окремій лекції.

На цьому ж етапі визначаються правила і процедури забезпечення цілісності БД.

3.1.5 Фізичне проектування

Етап фізичного проектування містить у собі реальне створення необхідних об'єктів БД (таблиць, індексів, обмежень і інше) відповідно до розробленої схеми БД. Це велика кропітка робота програмістів з використанням інструментальних засобів, що маються в конкретної СУБД.

3.2 Модель “сутність-зв'язок”

Модель “сутність-зв'язок” є найбільш розповсюдженим підходом до побудови БД. Її застосування стала діючим стандартом при інфологічному проектуванні БД. Загальноприйнятим стало скорочене найменування *ER*-модель (***Essence*** – сутність, ***Relation*** – зв'язок).

ER-модель добре співвідноситься з концепцією об'єктно-орієнтованого проектування і тому використовується в різних програмних засобах автоматизованого проектування систем (у так званих *CASE*-системах). Найбільш розповсюдженими засобами інфологічного проектування на даний час є, наприклад, програмні системи ERwin, POWER DESIGNER та ін.

Розглянемо коротко базові поняття *ER*-моделі (багато з них вам покажуться знайомими, тому що вони зв’язані з поняттями реляційної моделі даних).

Поняттям **сутність** представляється клас однотипних об'єктів. Передбачається, що існує множина екземплярів даної сутності, що представляють реальні об'єкти предметної області. Об'єкт, якому відповідає дана сутність, має свій набір **атрибутів** – характеристик, що описують властивості даного об'єкта. При цьому набір атрибутів повинний бути таким, щоб можна було розрізняти конкретні екземпляри даної сутності. Графічно сутність прийнято зображувати прямокутником, у верхній частині якого записується ім'я сутності, а нижче перелічуються всі атрибути даної сутності. На рис. 3.1 показаний приклад зображення сутності “Співробітник”.

Ключовими атрибутами називається такий набір атрибутів, що однозначно ідентифікує кожен екземпляр сутності. Наприклад, для сутності “Співробітник” це може бути атрибут “Табельний номер”. Ключові атрибути виділяються підкресленням чи іншим шрифтом.

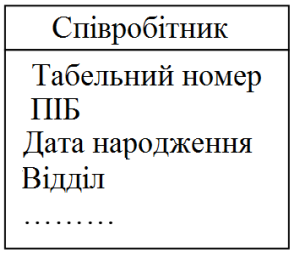


Рисунок 3.1 - Зображення сутності “Співробітник”

Між сутностями завжди встановлюються зв'язки, тому що всі об'єкти реального світу зв'язані між собою. Зв'язкам привласнюються найменування. Наприклад, між сутностями “Студент” і “Викладач” може існувати зв'язок “Читає лекції”. На схемі моделі зв'язки позначаються лініями, що з'єднують значки відповідних сутностей. Наприклад, так, як це показано на рис. 3.2.

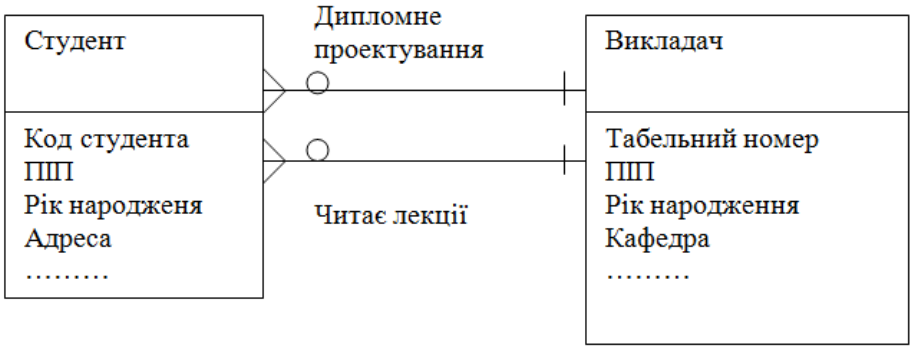


Рисунок 3.2 - Зображення зв'язку між сутностями

Зв'язки між сутностями можуть бути типів “один до одного” (1:1), “один до кількох” (1:M) чи “кілька до кількох” (M:M).

Зв'язок типу “один до одного” (1:1) означає, що один екземпляр першої сутності зв'язаний тільки з одним екземпляром другої сутності.

Зв'язок типу “один до кількох” (1:M) означає, що один екземпляр першої сутності може бути зв'язаний з декількома екземплярами другої сутності, але кожен екземпляр другої сутності - зв’язана тільки з одним екземпляром першої сутності.

Зв'язок типу “кілька до кількох” (M:M) означає, що один екземпляр першої сутності може бути зв'язаний з декількома екземплярами другої сутності і навпаки, кожен екземпляр другої сутності може бути зв'язаний з декількома екземплярами першої сутності.

У розглянутому вище прикладі зв'язок “Читає лекції” є зв'язком типу “один до кількох”, причому першою сутністю тут є “Викладач”, а другою – “Студент”.

У різних CASE-системах використовуються різні способи позначення типів зв'язків. Наприклад, у системі POWER DESIGNER прийнято зв'язок “один до кількох” позначати потрійною лінією з боку “кілька” (приблизно так, як це показано на рис. 3.2). Між двома сутностями може бути встановлено як завгодно багато зв'язків.

Наприклад, між сутностями “Студент” і “Викладач” ще може бути встановлений зв'язок “Дипломне проектування”. Очевидно, що цей зв'язок так само є зв'язком типу “один до кількох” – один викладач може керувати дипломною роботою декількох студентів, але кожен студент може мати тільки одного керівника дипломного проекту. Зв'язки будь-якого типу можуть бути обов'язковими чи необов'язковими.

Якщо в даному зв'язку повинен брати участь кожен екземпляр сутності, то зв'язок у даному випадку є обов'язковим, а якщо не кожен екземпляр – то необов'язковим. При цьому зв'язок

може бути обов'язковим з однієї сторони і необов'язковим з іншої сторони. Обов'язковий зв'язок на схемі прийнято позначати кружечком, а необов'язковий – перпендикулярною лінією, що перекреслює лінію зв'язку. На рис. 3.2 зв'язок "Дипломне проектування" з боку студента є обов'язковим, а з боку викладача – необов'язковим. Дійсно, кожен студент-дипломник повинен обов'язково мати керівника дипломного проекту, але не кожен викладач є керівником дипломного проекту.

3.3 Наслідування сутностей

Відповідно до принципів об'єктно-орієнтованого проектування для сутностей уводиться поняття типу і підтипу сутності. Це значить, що для будь-якої сутності ми можемо ввести її підтипи, у яких наслідуються всі атрибути старшої сутності, і ще вводяться додаткові атрибути, за допомогою яких визначаються специфічні властивості для даного підтипу сутності.

Пояснити ідею типів і підтипів можна на такому прикладі. Нехай, мається сутність "Факультет", у якій представлені атрибути, загальні для будь-яких цехів. В університеті є багато факультетів, що істотно розрізняються по спеціальності, напрямку підготовки, призначенню, складу кафедр і, отже, по атрибутах, за допомогою яких можна описувати їхні характеристики. У цьому випадку доцільно для сутності "Факультет" увести підтипи, у яких визначити необхідні специфічні атрибути. Наприклад, підтипами тут можуть бути: "Факультет ТК", "Факультет ІТ", "Факультет ІБї" і інше. На рис. 3.3 зображено приклад діаграми, що описує підтипи сутності "Факультет".

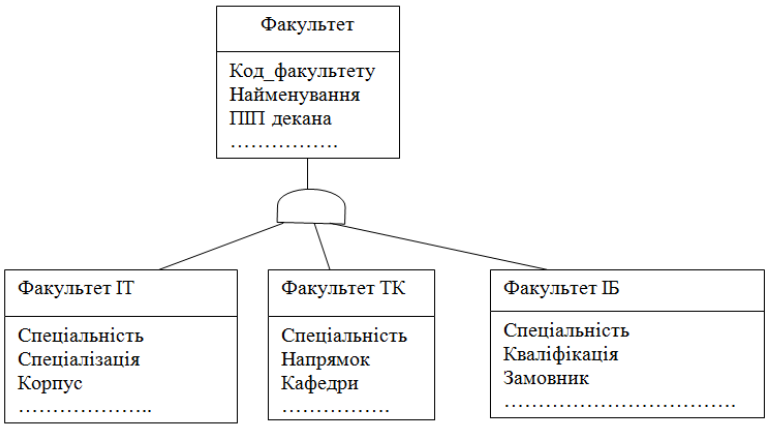


Рисунок 3.3 - Діаграма підтипів сутності "Факультет"

Сутність, на основі якої будуються підтипи, часто називають супертипом. На діаграмах для зображення зв'язку супертипу з підтипами сутностей використовується вузол-дискримінатор, що зображується у виді півкола, зверненого опуклою стороною до супертипу (як на рис. 3.3). Множина підтипів повинна бути вичерпною, тобто включати всі можливі варіанти представлення даної сутності в підтипах. Для цього, при необхідності, може вводитися підтип "Інші". У результаті розробки інформаційно-логічної моделі повинний бути отриманий зв'язний граф, у вузлах якого розташовуються введені сутності, зв'язані необхідними зв'язками. В отриманому графі не повинне бути циклічних зв'язків. Наявність циклічних зв'язків свідчить про некоректність моделі.

У завершення даного розділу розглянемо простий приклад моделі "сутність-зв'язок". Для прикладу візьмемо учбову БД, у якій необхідно зберігати інформацію про успішність студентів. Нехай у наслідок системного аналізу предметної області були визначені такі сутності: Група, Студент, Предмет, Викладач, Успішність.

Розроблена схема моделі "сутність-зв'язок" може бути такою, що зображена на рис. 3.4. Пояснимо найбільш важливі елементи й особливості цієї схеми.

Рисунок 3.4 - Граф-схема моделі “сутність-зв'язок” навчальної БД

Між сутностями “Група” і “Студент” мається зв'язок “Належить”. Зв'язок є обов'язковим і тип зв'язку – “один до кількох”. У кожну групу повинні обов'язково входити більш одного студента. Кожен студент обов'язково входить до складу якої-небудь однієї групи.

У сутності “Успішність” представляються оцінки, отримані студентами на заняттях у різний час. Зв'язок між сутностями “Студент” і “Успішність” має назву “Бере участь на заняттях”. Зв'язок має тип “1:M”, тому що один студент може мати кілька оцінок чи ні однієї. Тому зв'язок є обов'язкової з боку сутності “Успішність” і необов'язкової з боку сутності “Студент”.

Зв'язок між сутностями “Предмет” і “Успішність” може мати назву “Заняття по предметах”. Тип зв'язку – “1:M”. По кожному предмету можуть бути отримані ні однієї чи кілька оцінок, але кожна оцінка завжди відноситься до якого-небудь одного предмету.

Зв'язок між сутностями “Викладач” і “Предмет” має назву “Викладає”. Тип зв'язку можна вважати “1:M” з урахуванням того, що кожен предмет веде тільки один викладач (якщо викладачів два, то основний завжди один). Зв'язок є обов'язковим.

Первинні ключі сутностей виділено напівжирним шрифтом.

3.4 Перехід до реляційної моделі

Модель “сутність-зв'язок” перетворюється у реляційну модель відповідно до наступних правил:

1. Кожної сутності ставиться у відповідність **відношення** реляційної моделі. Імена сутностей і відношень можуть відрізнятися. При цьому враховуються вимоги обраної СУБД до імен відношень (таблиць).
2. Кожен атрибут сутності стає атрибутом відповідного відношення. Імена атрибутів повинні бути короткими і задовольняти вимогам до імен атрибутів (полів) обраної СУБД. Наприклад, необхідно ураховувати, що на багатьох серверах БД заборонені пробіли в іменах атрибутів. Тому рекомендується не використовувати пробіли, заміняючи їх символом “підкреслення”. Для кожного атрибута відносини визначається **тип даних**, вказується, **обов'язковим** чи **необов'язковим** є цей атрибут.
3. Первинний ключ сутності стає **первинним ключем відношення.**

4. У кожному відношенні, яке представляє відповідну підлеглу сутність (в зв'язку типу “1:M”), додається набір тих атрибутів з основної сутності, які у неї є первинним ключем. У відношенні, що відповідає підлеглої сутності, цей набір атрибутів стає **зовнішнім ключем**. Відповідний зв'язок “1:M” позначається умовою рівності значень цих атрибутів, які тепер стали зовнішнім ключем.

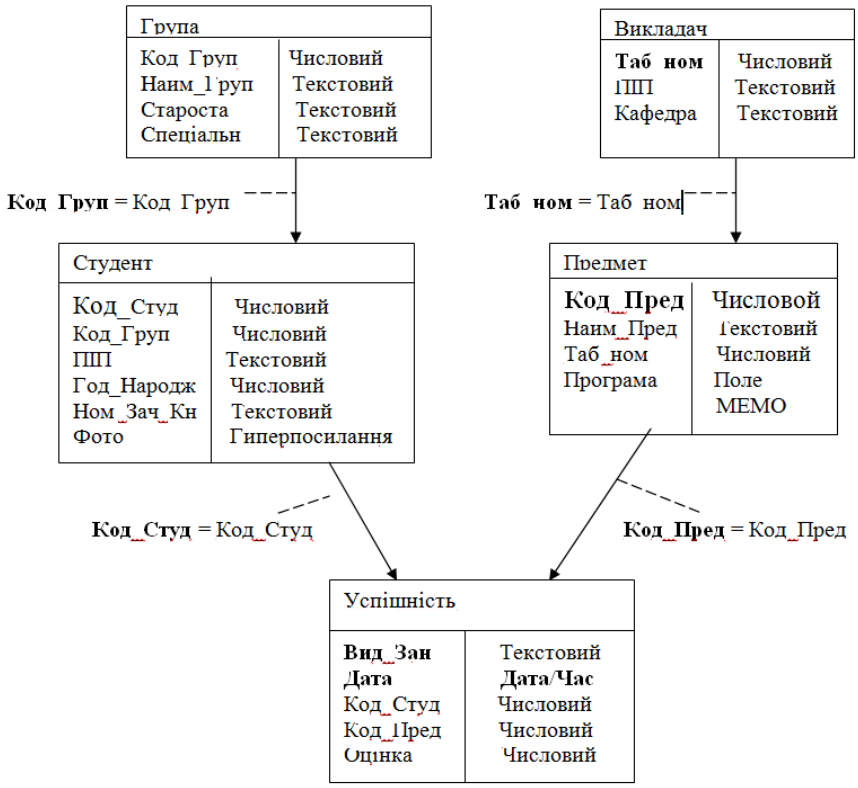


Рисунок 3.5 - Схема відносин реляційної БД

На рис. 3.5 приведено схему відношень, яка отримана після перетворення моделі “сутність-зв'язок”, граф якої був побудований вище і зображений на рис. 3.4. Первинні ключі на схемі відношень позначені напівжирним шрифтом. На схемі відносин типи даних зазначені в припущенні, що в якості СУБД обрана MS Access.

3.5 Застосування CASE-засобу AllFusion ERwin Data Modeler для проектування БД

В умовах ринку усе більше число компаній усвідомлюють переваги використання ІС. У деяких випадках ІС - це не тільки набір послуг, але і найважливіший компонент бізнесу, як, наприклад, система резервування квитків або засобу надання фінансової інформації. Щоб одержати вигоду від використання інформаційної системи, неї варто створювати в короткий термін і зі зменшеними витратами. Інформаційна система повинна бути легко супроводжуваної і керованої. Створення інформаційної системи підприємства - досить складний і багатоступінчастий процес, що, досить часто, містить фазу інформаційного моделювання.

Інформаційна модель - це специфікація структури даних і бізнес правил (правил предметної області).

3.5.1 CASE-технології проектування БД

CASE (англ. Computer-Aided Software Engineering) — набір інструментів і методів програмної інженерії для проектування програмного забезпечення, що допомагає забезпечити високу якість програм, відсутність помилок і простоту в обслуговуванні програмних продуктів.

Також під CASE розуміють сукупність методів і засобів проектування інформаційних систем з інтегрованими автоматизованими інструментами, які можуть бути використані в процесі розробки програмного забезпечення.

До функцій CASE входять засоби аналізу, проектування й програмування. За допомогою CASE автоматизують процеси проектування інтерфейсів, документування й генерування структурованого коду бажаною мовою програмування.

Типовими CASE інструментами є:

- інструменти управління конфігурацією;
- інструменти моделювання даних;
- інструменти аналізу й проектування;
- інструменти перетворення моделей;
- інструменти редагування програмного коду;
- інструменти рефакторингу коду;
- генератори коду;
- інструменти для побудови UML-діаграм.

Розглянемо деякі аспекти інформаційного моделювання і його автоматизації з використанням CASE-засобу AllFusion ERwin Data Modeler (рис.3.6).

ERwin — засіб розробки структури бази даних (БД). ERwin сполучить графічний інтерфейс Windows, інструменти для побудови ER-діаграм, редактори для створення логічного і фізичного опису моделі даних і прозору підтримку ведучих реляційних СУБД і настільних баз даних.

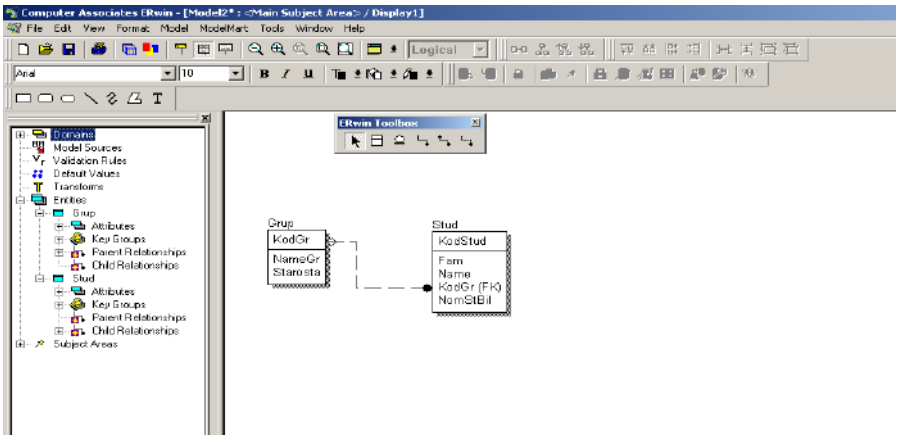


Рисунок 3.6 - Інтерфейс програми ERwin

У AllFusion ERwin Data Modeler реалізований ряд функцій:

- пряме підключення до бази даних: створення структури бази даних безпосередньо з ERwin, відновлення моделі існуючої БД;
- перехід від однієї цільової бази даних до іншої з використанням взаємно однозначних відповідності особливостей СУБД;
- підтримка «настільних» (desktop) СУБД;
- керування фізичними характеристиками збереження даних (для Oracle і DB2 — табличним простором і сегментами відповідно);
- розбивка діаграми на функціонально закінчені частини — логічні області;
- збережені набори параметрів відображення для побудови звітів і діаграм;
- процедури і тригери описуються при побудові моделі й автоматично створюються в БД при генерації;
- технологія «drag and drop» для маніпулювання атрибутами;
- можливість збереження діаграми в цільовій базі даних або в DBF файлах;
- підтримка системи контролю версій PVCS фірми Intersolv;
- шрифтове і колірне виділення.

Реалізація моделювання в ERwin базується на теорії реляційних БД і на методології IDEF1X.

Методологія IDEF1X була розроблена для ВВС США і тепер використовується, зокрема, в урядових, аерокосмічних і фінансових установах, а також у великому числі приватних компаній.

Вона визначає стандарти термінології, використовуваної при інформаційному моделюванні, і графічного зображення типових елементів на діаграмах.

Можливі дві точки зору на інформаційну модель і, відповідно, два рівні моделі. Перший — логічний (точка зору користувача) — описує дані, задіяні в бізнесі підприємства. Другий — фізичний — визначає представлення інформації в БД. ERwin поєднує них у єдину діаграму, що має кілька рівнів представлення.

Процес побудови інформаційної моделі складається з наступних кроків:

- визначення сутностей;
- визначення залежностей між сутностями;
- завдання первинних і альтернативних ключів;
- визначення атрибутів сутностей;
- приведення моделі до необхідного рівня нормальної форми;
- перехід до фізичного опису моделі: призначення відповідностей ім'я сутності - ім'я таблиці, атрибут сутності - атрибут таблиці; завдання тригерів, процедур і обмежень;
- генерація бази даних.

ERwin створює візуальне представлення (модель даних) для розв'язуваної задачі. Це представлення може використовуватися для детального аналізу, уточнення і поширення як частини документації, необхідної в циклі розробки. Однак ERwin далеко не тільки інструмент для малювання. ERwin автоматично створює базу даних (таблиці, індекси, збережені процедури, тригери для забезпечення посилальної цілісності й інші об'єкти, необхідні для керування даними).

Для створення нової моделі у головному меню потрібно вибрати: **File -> New.** Відкриється вікно із списком моделей (рис.3.7), які можна створити.

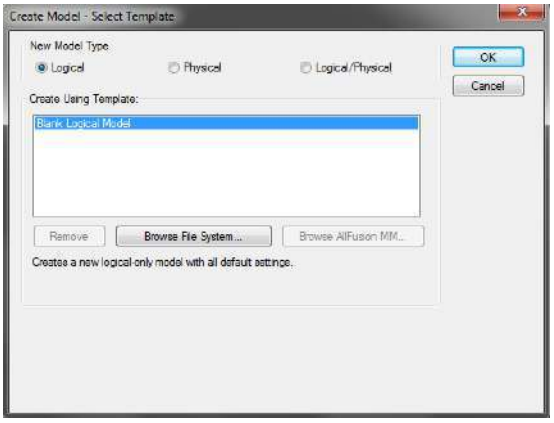


Рисунок 3.7 - Вибір типу моделі

Якщо вибрати фізичну або логіко-фізичну модель, то можна також обрати цільову СУБД, наприклад, MySQL (рис.3.8).

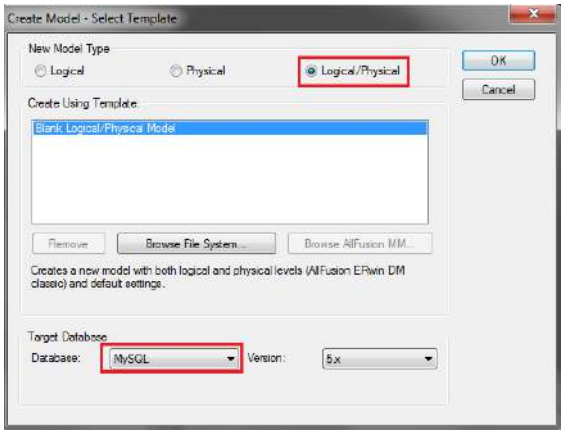


Рисунок 3.8 - Вибір СУБД

Після вибору вказаних опцій і натискання «OK».

Для конфігурування моделі на дереві елементів клацаємо правою кнопкою миші по **Model** і обираємо **Properties** (рис.3.9).

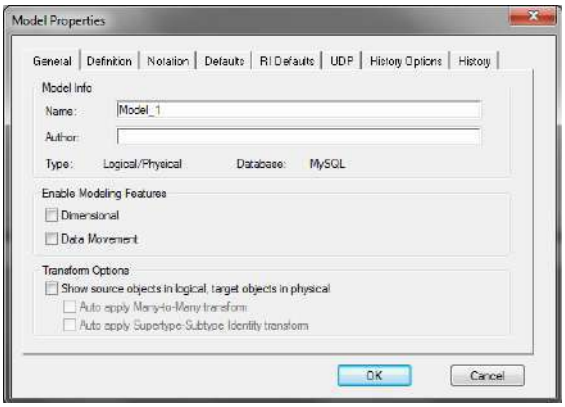


Рисунок 3.9 - Конфігурування моделі

У цьому вікні можна за потреби обрати створення багатовимірної моделі (**Dimensional**; використовується для проектування сховищ даних) і вибрати нотацію у вкладці **Notation** (рис. 3.10).

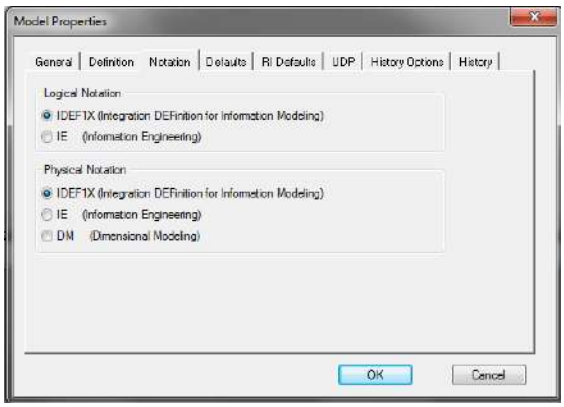


Рисунок 3.10 - Вибір нотації графічного зображення моделі

Після цього, використовуючи меню швидкого доступу до елементів проектування, можна додавати в модель сутності, зв’язки між ними та інші об’єкти (рис.3.11).

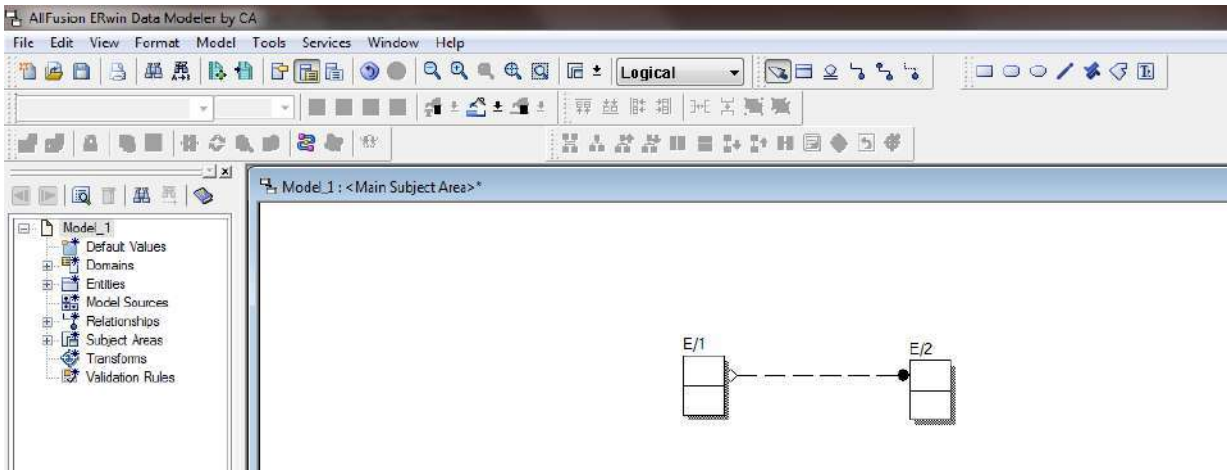


Рисунок 3.11 - Бланк нової моделі БД

Використовуючи меню **Format** можна обирати рівні представлення моделі (**Display Level**), а також відображати/прибирати різні елементи моделі (**Entity Display** та **Relationship Display**).

За допомогою ERwin можна створювати або проводити зворотне проектування (реінжиніринг) баз даних.

3.5.2. Режими відображення рівня моделі даних у ERwin

У ERwin існують два рівні представлення і моделювання — логічний і фізичний. Логічний рівень означає пряме відображення фактів з реального життя. Наприклад, люди, столи, відділи, комп'ютери є реальними об'єктами. Вони іменуються природною мовою, з будь-якими роздільниками слів (пробіли, коми і інше).

На логічному рівні не розглядається використання конкретної СУБД, не визначаються типи даних (наприклад, ціле або речовинне число) і не визначаються індекси для таблиць (рис.3.12).

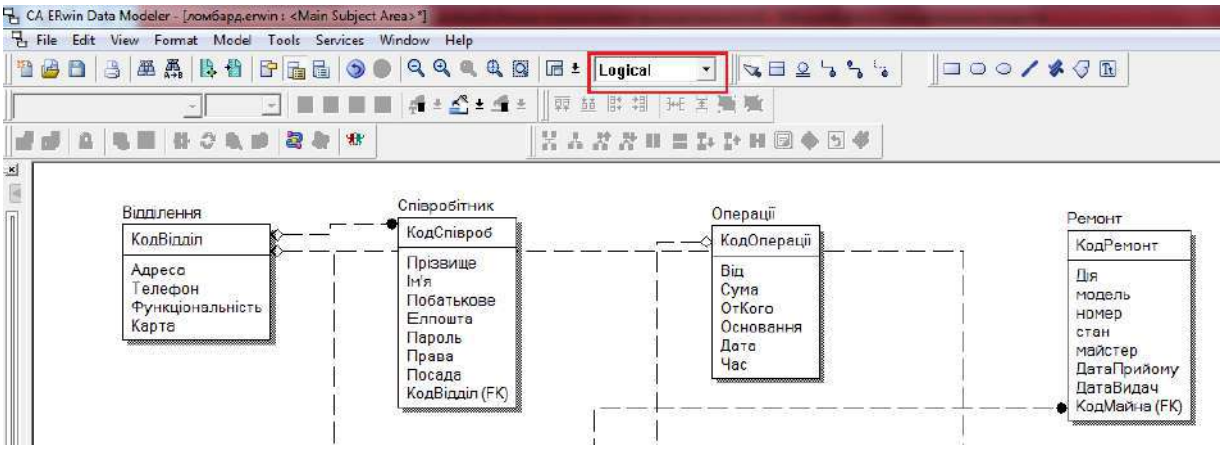


Рисунок3.12 - Відображення на логічному рівні моделі БД

Цільова СУБД, імена об'єктів і типи даних, індекси складають другий (фізичний) рівень моделі ERwin (рис. 3.13).

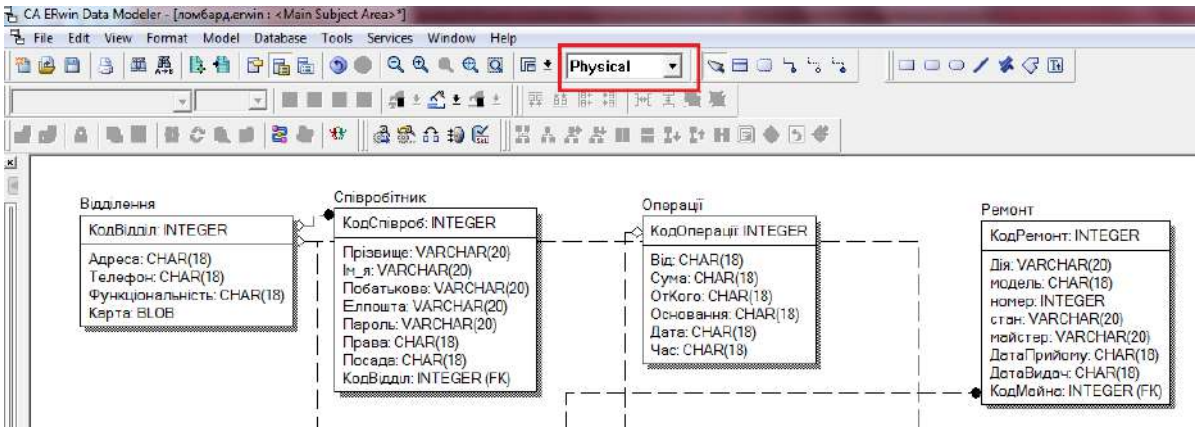


Рисунок 3.13 - Відображення на фізичному рівні моделі БД

Для налаштування фізичного рівня зображення моделі потрібно відкрити вкладку Format, вибрати Table Display та встановити Column Datatype (рис. 3.14).

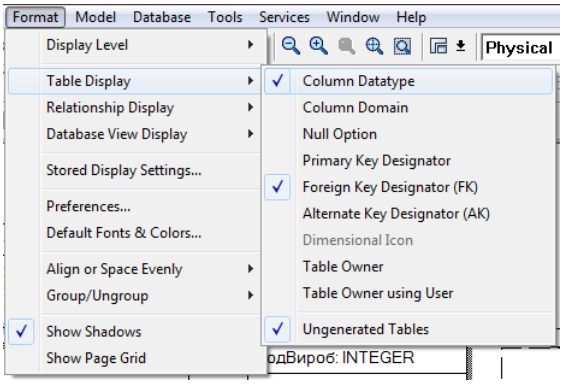


Рисунок 3.14 - Налаштування зображення на фізичному рівні моделі БД

ERwin надає можливості створювати і керувати цими двома різними рівнями представлення однієї діаграми (моделі), так само як і мати багато варіантів відображення на кожнім рівні.

Діаграма ERwin будується з трьох основних блоків — сутностей, атрибутів і зв'язків. Якщо розглядати діаграму як графічне представлення правил предметної області, то сутності є іменниками, а зв'язку - дієсловами.

Вибір між логічним і фізичним рівнем відображення здійснюється через лінійку інструментів або меню. Усередині кожного з цих рівнів є наступні режими відображення:

- **Режим «сутності»** - усередині прямокутників відображається ім'я сутності (для логічної моделі) або ім'я таблиці (для фізичного представлення моделі); служить для зручності огляду великої діаграми або розміщення прямокутників сутностей на діаграмі (рис.3.15).

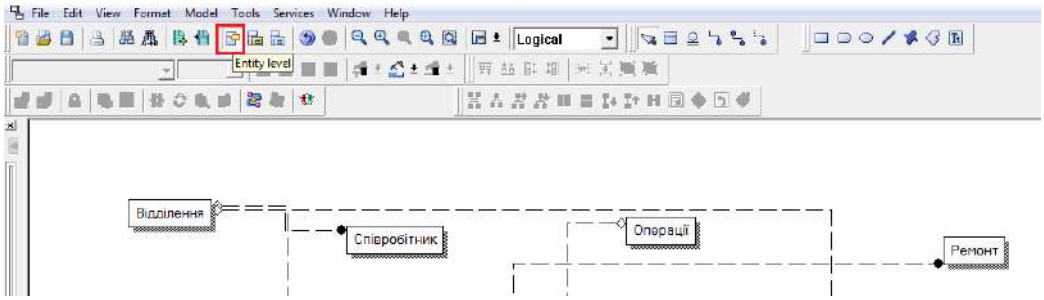


Рисунок 3.15 - Відображення моделі БД на рівні «сутності»

- **Режим «визначення сутності»** служить для презентації діаграми іншим людям (рис.3.16).

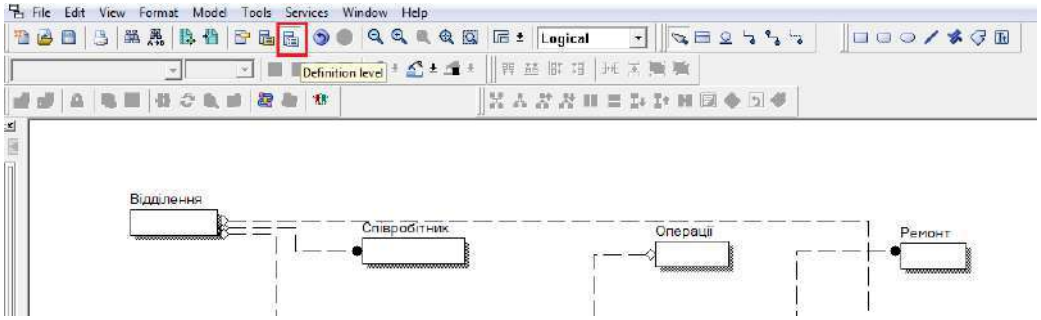


Рисунок 3.16. Відображення моделі БД на рівні «визначення сутності»

- **Режим «атрибути».** При переході від предметної області до моделі потрібно вводити інформацію про те, що складає сутність (рис.3.12). Ця інформація вводиться шляхом завдання атрибутів (на фізичному рівні — стовпчиків таблиць). У цьому режимі прямокутник-сутність поділяється лінією на двох частин — у верхній частині відображаються атрибути (стовпчика), що складають первинний ключ, а в нижньої — інші атрибути. Цей режим є основним при проектуванні на логічному і фізичному рівнях.

Діаграма може займати більш ніж один екран і більш ніж один лист при печатці. Для огляду моделі передбачені, крім прокручувань екрана, режими зменшення зображення в два і чотири рази.

3.5.3. Інструменти для створення моделі в ERwin

Основні інструменти створення моделі доступні як з меню, так і через вікно інструментів (рис.3.17). З їхньою допомогою створюються незалежні і залежні сутності, що ідентифікують і не ідентифікують зв'язки, повні і неповні категорії, неспецифічні зв'язки і текстові елементи.

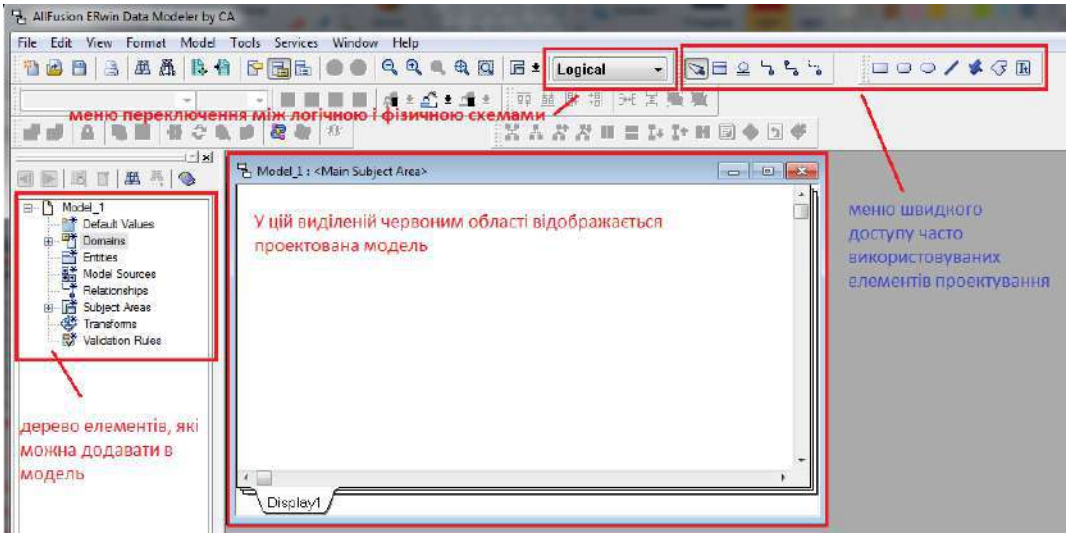


Рисунок 3.17. Панелі інструментов інтерфейсу ERwin

Натисканням миші над сутністю виробляється вхід в один з численних редакторів ERwin:

- редактори, зв'язані із сутністю в цілому (визначення сутності, додаткова інформація, тригери, індекси, характеристики таблиці, збережені процедури, зв'язані з таблицею);
- редактори атрибутів (визначення атрибутів, стовпчика таблиці у фізичному представленні моделі, репозитарій засобу 4GL, наприклад, розширені атрибути в PowerBuilder. Для додовання нового атрибуту потрібно правою клавишою миши визвать контекстне меню та вибрать Attributes (рис.3.18).

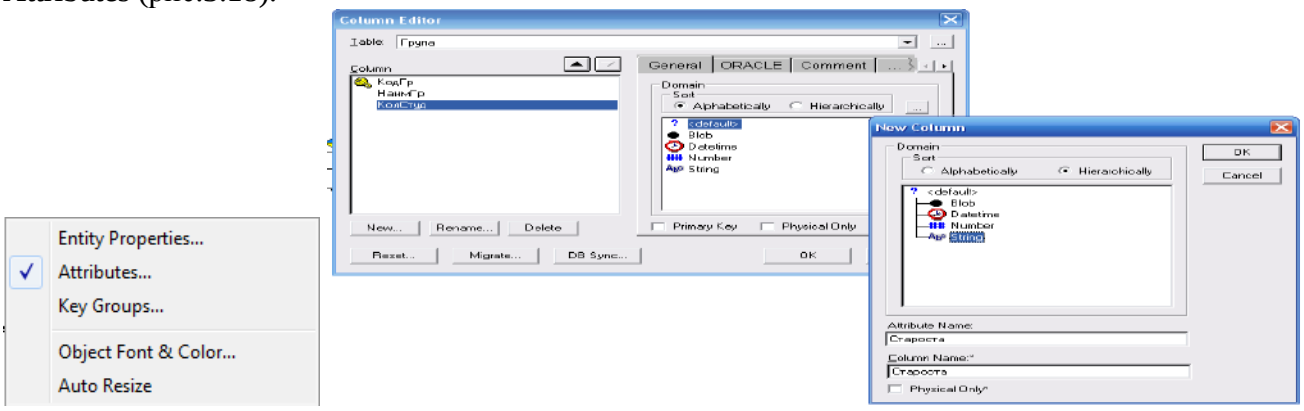


Рисунок 3.18 - Редактор атрибутів

На діаграмі сутність зображується прямокутником. У залежності від режиму представлення діаграми прямокутник може містити ім'я сутності, її опис, список її атрибутів і інші зведення.

Горизонтальна лінія прямокутника розділяє атрибути сутності на два набори — атрибути, що складають первинний ключ у верхній частині, та інші (не вхідні в первинних ключ) — у нижній частині.

Сутність являє собою безліч реальних або абстрактних об'єктів, наприклад: люди, місця, події, факти, що мають загальні характеристики. Сутність — це логічне поняття. Сутності відповідає таблиця в реальної СУБД. У ERwin сутність візуально представляє три основних види інформації:

- атрибути, що складають первинний ключ;
- не ключові атрибути;
- тип сутності (незалежна/залежна).

Первинний ключ — це атрибут або набір атрибутів, унікально ідентифікуючий екземпляр сутності. Якщо кілька наборів атрибутів можуть унікально ідентифікувати сутність, то вибір одного з них здійснюється розроблювачем на підставі аналізу предметної області.

Для кожного первинного ключа ERwin створює при генерації структури БД унікальний індекс.

Екземпляри незалежної сутності можуть бути унікально ідентифіковані без визначення її зв'язків з іншими сутностями; залежна сутність, навпаки, не може бути унікально ідентифікована без визначення її зв'язків з іншими сутностями. Залежна сутність відображається в ERwin прямокутником із закругленими кутами.

Зв'язок - це функціональна залежність між двома сутностями (зокрема, можливий зв'язок сутності із самої собою). Наприклад, важливо знати прізвище співробітника, і не менш важливо знати, у якому відділі він працює. Таким чином, між сутностями «відділ» і «співробітник» існує зв’язок який «складається з» (відділ складається зі співробітників). Зв'язок — це поняття логічного рівня, якому відповідає зовнішній ключ на фізичному рівні.

У ERwin зв'язки представлені п'ятьма основними елементами інформації:

- тип зв'язку (ідентифікуюча, не ідентифікуюча, повна/неповна категорія, неспецифічний зв'язок);
- батьківська сутність;
- дочірня (залежна) сутність;
- потужність зв'язку (cardinality);
- допустимість порожніх (null) значень.

Зв'язок називається ідентифікуючої , якщо екземпляр дочірньої сутності ідентифікується через її зв'язок з батьківською сутністю (рис.319).

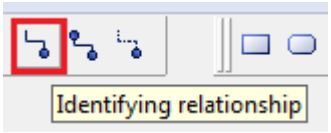


Рисунок 3.19 - Кнопка вибори ідентифікуючого звязку

Атрибути, що складають первинний ключ батьківської сутності, при цьому входять у первинний ключ дочірньої сутності. Дочірня сутність при ідентифікуючому зв'язку завжди є залежною (рис.3.20).

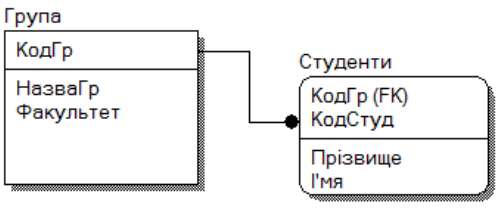


Рисунок 3.20 Модель з ідентифікуючім типом зв’язку

Зв'язок називається не ідентифікуючої, якщо екземпляр дочірньої сутності ідентифікується інакше, чим через зв'язок з батьківською сутністю (рис.3.21).

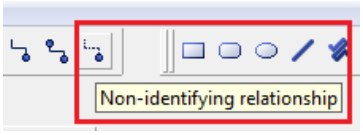


Рисунок 3.21 Кнопка вибори неідентифікуючого зв’язку

Атрибути, що складають первинний ключ батьківської сутності, при цьому входять до складу не ключових атрибутів дочірньої сутності (рис.3.22).

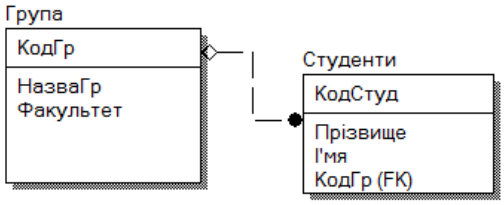


Рисунок 3.22 Модель з неідентифікуючім типом зв’язку

Для визначення зв'язків ERwin вибирається тип зв'язку, потім мишею вказується батьківська і дочірня сутність.

Ідентифікуючий зв'язок зображується суцільною лінією; не ідентифікуюча — пунктирною лінією. Лінії закінчуються крапкою з боку дочірньої сутності.

При визначенні зв'язку відбувається міграція атрибутів первинного ключа батьківської сутності у відповідну область атрибутів дочірньої сутності. Тому такі атрибути не уводяться вручну. Кожній зв'язок налаштовується за допомогою майстра, якій можна покликати, якщо навести маркер на лінію зв’язку, натиснуть праву кнопку миші та вибрати Relatioship (рис.3.23).

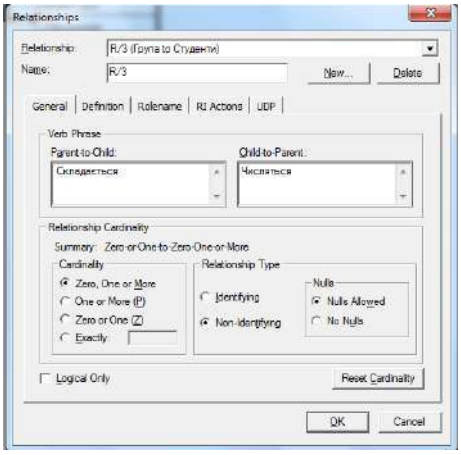


Рисунок 3.23. Вікно налаштування зв’язку

Ім'я зв'язку на логічному рівні являє собою «дієслово», що зв'язує сутності- Verb Phrase. Фізичне ім'я зв'язку (яке може відрізнятися від логічного) для ERwin означає ім'я обмеження (constraint) або індексу.

Атрибути первинного ключа батьківської сутності за замовчуванням мігрують зі своїми іменами. ERwin дозволяє ввести для них ролі, тобто нові імена, під якими мігруючі атрибути будуть представлені в дочірній сутності. У випадку кількаразової міграції атрибута таке перейменування необхідне. Наприклад, сутність «посередницька угода» має атрибут «код підприємства-продавця» і «код підприємства-покупця». У даному випадку первинний ключ сутності «підприємство» («код підприємства») має двох ролей у дочірній сутності.

На фізичному рівні ім'я ролі — це ім'я стовпчика зовнішнього ключа в дочірній таблиці.

Потужність зв'язку (Relationship Cardinality) являє собою відношення кількості екземплярів батьківської сутності до відповідної кількості екземплярів дочірньої сутності. Для будь-якого зв'язку, крім неспецифічної, цей зв'язок записується як l:n.

ERwin, відповідно до методології IDEF1X, надає 4 варіанти для n, що зображуються додатковим символом у дочірньої сутності:

- нуль, один або більше (за замовчуванням);
- один або більше (зображується буквою «P»);
- нуль або один (зображується буквою «Z»);
- рівно N, де N — конкретне число (зображується числом N).

Допустимість порожніх (NULL) значень у не ідентифікуючих зв'язках ERwin зображує порожнім ромбиком на дузі зв'язку з боку батьківської сутності.

Всі об'єкти моделі ERwin можуть редагуватися засобами, прийнятими в Windows, — угруповання, копіювання, видалення, переміщення, використання системного буфера. Установка квітів і шрифтів здійснюється в зручних діалогах.

Альтернативний ключ - це атрибут (або група атрибутів), незбіжний з первинним ключем і унікально ідентифікуючий екземпляр сутності. Наприклад, для сутності службовець (ідентифікатор що служить, прізвище, ім'я, по батькові) група атрибутів «прізвище», «ім'я», «по батькові» можуть бути альтернативним ключем.

Для альтернативного ключа, як і для первинного, ERwin автоматично створює індекси при генерації БД. Атрибути, що складають альтернативний ключ, однозначно (унікально) ідентифікують екземпляри сутності. У ERwin можна також складати групи атрибутів, що не ідентифікують унікально екземпляри сутності, але часто використовуються для доступу до даних. Для кожної такої групи атрибутів ERwin створює не унікальні індекси.

Ті самі атрибути сутності можуть входити в кілька різних груп ключів.

- тип зв'язку (ідентифікуюча, не ідентифікуюча, повна/неповна категорія, неспецифічний зв'язок);
- батьківська сутність;
- дочірня (залежна) сутність;
- потужність зв'язку (cardinality);
- допустимість порожніх (null) значень.

Зв'язок називається ідентифікуючої, якщо екземпляр дочірньої сутності ідентифікується через її зв'язок з батьківською сутністю. Атрибути, що складають первинний ключ батьківської сутності, при цьому входять у первинний ключ дочірньої сутності. Дочірня сутність при ідентифікуючому зв'язку завжди є залежною.

Зв'язок називається не ідентифікуючої, якщо екземпляр дочірньої сутності ідентифікується інакше, чим через зв'язок з батьківською сутністю. Атрибути, що складають первинний ключ батьківської сутності, при цьому входять до складу не ключових атрибутів дочірньої сутності.

Для визначення зв'язків ERwin вибирається тип зв'язку, потім мишею вказується батьківська і дочірня сутність. Ідентифікуючий зв'язок зображується суцільною лінією; не ідентифікуюча — пунктирною лінією. Лінії закінчуються крапкою з боку дочірньої сутності.

При визначенні зв'язку відбувається міграція атрибутів первинного ключа батьківської сутності у відповідну область атрибутів дочірньої сутності. Тому такі атрибути не уводяться вручну.

Атрибути первинного ключа батьківської сутності за замовчуванням мігрують зі своїми іменами. ERwin дозволяє ввести для них ролі, тобто нові імена, під якими мігруючі атрибути будуть представлені в дочірній сутності. У випадку кількаразової міграції атрибута таке перейменування необхідне. Наприклад, сутність «посередницька угода» має атрибут «код підприємства-продавця» і «код підприємства-покупця». У даному випадку первинний ключ сутності «підприємство» («код підприємства») має двох ролей у дочірній сутності.

На фізичному рівні ім'я ролі — це ім'я стовпчика зовнішнього ключа в дочірній таблиці.

Допустимість порожніх (NULL) значень у не ідентифікуючих зв'язках ERwin зображує порожнім ромбиком на дузі зв'язку з боку батьківської сутності.

Ім'я зв'язку на логічному рівні являє собою «дієслово», що зв'язує сутності. Фізичне ім'я зв'язку (яке може відрізнятися від логічного) для ERwin означає ім'я обмеження (constraint) або індексу.

Всі об'єкти моделі ERwin можуть редагуватися засобами, прийнятими в Windows, — угруповання, копіювання, видалення, переміщення, використання системного буфера. Установка квітів і шрифтів здійснюється в зручних діалогах.

Деякі сутності визначають категорію об'єктів одного типу. У ERwin у такому випадку створюється сутність для визначення категорії і для кожного елемента категорії, а потім уводиться для них зв'язок катетеризації. Батьківська сутність категорії називається супертипом, а дочірні — підтипом. Наприклад, сутність «співробітник» може містити дані як про штатних працівників, так і про тимчасово найняті. Перші і другі мають різні, частково пересічні набори атрибутів (мінімальне перетинання підтипів складає первинний ключ). Загальна частина цих атрибутів, включаючи первинний ключ, міститься в сутність - супертип «співробітник».

Різна частина (наприклад, дані погодинної оплати для тимчасових працівників і дані про зарплату і відпустку для штатних працівників) міститься в сутності-підтипи.

У сутності супертипе вводиться атрибут-дискримінатор, що дозволяє розрізняти конкретні екземпляри сутності — підтипу.

У залежності від того, чи всі можливі сутності-підтипи включені в модель, категорійний зв'язок є повною або неповною. Продовжуючи приклад, якщо супертип може містити дані про звільнених співробітників, те цей зв'язок - неповної катетеризації, тому що для нього не існує запису в сутностях - підтипах. У ERwin повна категорія зображується окружністю з двома підкресленнями, а неповна — окружністю з одним підкресленням.

Посилальна цілісність — це забезпечення вимоги, щоб значення зовнішнього ключа екземпляра дочірньої сутності відповідали значенням первинного ключа в батьківській сутності. Посилальна цілісність може контролюватися при всіх операціях, що змінюють дані (INSERT/UPDATE/DELETE). Засобу контролю посилальної цілісності в ERwin включають автоматичну генерацію тригерів і використання механізмів декларативної посилальної цілісності (для тих СУБД, що підтримують дані механізми).

Для кожного зв'язку на логічному рівні можуть бути задані вимоги по обробці операцій INSERT/UPDATE/DELETE для батьківської і дочірньої сутності. ERwin представляє наступні варіанти обробки цих подій:

- відсутність перевірки;
- перевірка допустимості;
- заборона операції;
- каскадне виконання операції (DELETE/UPDATE);
- установка порожнього (NULL-значення) або заданого значення за замовчуванням.

Відповідно до обраного варіанта ERwin автоматично створює необхідні тригери на діалекті SQL цільовий СУБД. При цьому ERwin користується бібліотекою шаблонів тригерів, які можна модифікувати.

Звичайно моделі ERwin зберігаються на диску у виді файлу. Мається можливість зберігати модель у цільовий СУБД. Для цього за допомогою самого ERwin у цільовий СУБД створюється позначка-база ERwin. У цій базі даних зберігається інформація моделі. В окремому випадку базою даних можуть бути і dBase-файли, з якими ERwin працює через ODBC.

3.6 Проектування сховищ даних за допомогою CASE системи ERwin

Сховища даних (Data Warehouse) являють собою спеціалізовані бази даних, призначені для збереження даних, що рідко міняються, але на основі яких часто потрібне виконання складних запитів. Звичайно вони орієнтовані на виконання аналітичних запитів, що забезпечують підтримку прийняття рішень для керівників і менеджерів. Сховища даних дозволяють розвантажити промислові бази даних, і тим самим, дозволяють користувачам більш ефективно і швидко витягати необхідну інформацію. Як правило, сховища даних оперують з величезними обсягами інформації, що пред'являє до їх проектування і реалізації підвищені вимоги. Вибір як платформу сховища даних такий високопродуктивних РСУБД дозволяє істотно підвищити загальну ефективність створюваної інформаційної системи. У цьому випадку ERwin (CASE-засіб фірми PLATINUM technology, inc.) стає незамінним інструментом, оскільки з однієї сторони ефективно підтримує на фізичному рівні проектування об'єктів РСУБД, з іншої сторони має спеціалізовані засоби моделювання сховищ даних.

Нижче розглядаються основні можливості ERwin по проектуванню сховищ даних.

До проектування сховищ даних звичайно пред'являються наступні вимоги:

- Структура дані сховища повинна бути зрозуміла користувачам;
- Повинні бути виділені статичні дані, що регулярно модифікуються: щодня, щотижня і ін.;
- Повинні бути спрощені вимоги до запитів з метою виключення запитів, що могли б вимагати множинних тверджень SQL у традиційних реляційних СУБД;
- Повинна бути забезпечена підтримка складних запитів SQL, що вимагають послідовної обробки тисяч або мільйонів записів.

Ці вимоги істотно відрізняють структуру реляційних СУБД і сховищ даних. Нормалізація даних у реляційних СУБД приводить до створення безлічі зв'язаних між собою таблиць. У результаті, виконання складних запитів неминуче приводить до об'єднання багатьох таблиць, що істотно збільшує час відгуку. Проектування сховища даних має на увазі створення денормалізованої структури даних (допускається надмірність даних і можливість виникнення аномалій при маніпулюванні даними), орієнтованої в першу чергу на високу продуктивність при виконанні аналітичних запитів. Нормалізація робить модель сховища занадто складної, утрудняє її розуміння і погіршує ефективність виконання запиту.

Для ефективного проектування сховищ даних ERwin використовує розмірну (Dimensional) модель. Dimensional - методологія проектування спеціально призначена для розробки сховищ даних. ERwin підтримує методологію розмірного моделювання завдяки використанню спеціальної нотації для фізичної моделі – Dimensional. Найбільш простий спосіб перейти до нотації Dimensional - при створенні нової моделі (меню File / New) у діалозі ERwin Teamplate Selection вибрати зі списку пропонованих шаблонів DIMENSION. У шаблоні DIMENSION зроблені всі необхідні для підтримки нотації розмірного моделювання настроювання, що, утім, можна установити вручну.

Моделювання Dimensional подібно з моделюванням зв'язків і сутностей для реляційної моделі, але відрізняються, цілями. Реляційна модель акцентується на цілісності й ефективності введення даних. Розмірна (Dimensional) модель орієнтована в першу чергу на виконання складних запитів до БД.

У розмірному моделюванні прийнятий стандарт моделі, називаний схемою зірка (star schema), що забезпечує високу швидкість виконання запиту, за допомогою денормалізації і поділу даних. Неможливо створити універсальну денормалізовану структуру даних, що забезпечує високу продуктивність при виконанні будь-якого аналітичного запиту. Тому схема зірка будується так, щоб забезпечити найвищу продуктивність при виконанні одного найважливішого запиту, або для групи схожих запитів.

Схема зірка звичайно містить одну велику таблицю, називану таблицею факту (fact table), поміщену в центр, і навколишні її менші таблиці, називані таблицями розмірності (dimensional table), з'єднаними с таблицею факту у виді зірки радіальними зв'язками. У цих зв'язках таблиці розмірності є батьківськими, таблиця факту - дочірньої. Схема зірка може мати консольні таблиці (outrigger table), приєднані до таблиці розмірності. Консольні таблиці є батьківськими, таблиці розмірності - дочірніми.

У розмірній моделі, ERwin позначає іконкою роль таблиці в схемі зірка:


Таблиця факту (fact table)


Таблиця розмірності (dimensional table),


Консольна таблиця (outrigger table).

Перш ніж створити базу даних зі схемою типу зірка, необхідно проаналізувати бізнес-правила предметної області з метою з'ясування центрального питання, відповідь на який найбільш важливий. Всі інші питання повинні бути об'єднані навколо цього основного питання і моделювання повинне починатися з цього основного питання. Дані, необхідні для відповіді на це питання, повинні бути поміщені в центральну таблицю моделі - таблицю факту. Наприклад, якщо необхідно створювати звіти про загальну суму доходу від продажів за період або по типі товару або по продавцях, варто розробляти модель так, щоб кожен запис у таблиці факту представляв загальну суму продажів, для кожного клієнта за визначений період часу для кожного продавця (рис.3.24).

У прикладі, таблиця факту містить сумарні дані про продажі («SALE»), а таблиці розмірності містять дані про замовника і замовлення («CUSTOMER»), продуктах («PRODUCT»), продавцях («SALESPEOPLE») і періодах часу («TIME»).

Таблиця факту є центральною таблицею в схемі зірка. Вона може складатися з мільйонів рядків і містити підсумовуючі або фактичні дані, що можуть допомогти відповісти на необхідні питання. Вона з'єднує дані, що зберігалися б у багатьох таблицях традиційних реляційних базах даних. Таблиця факту і таблиці розмірності зв'язані ідентифікуючими зв'язками, при цьому первинні ключі таблиці розмірності мігрують у таблицю факту як зовнішні ключі.

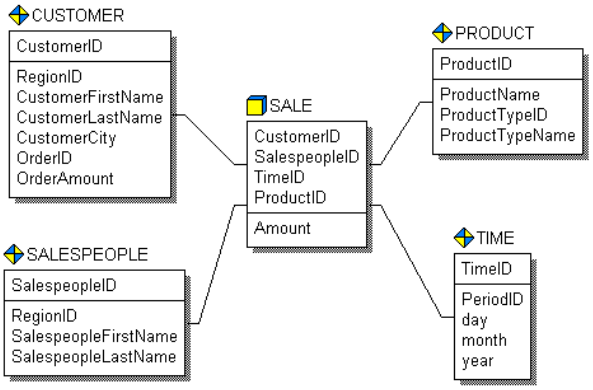


Рисунок 3.24 - Схема зірка.

У розмірній моделі напрямку зв'язків явно не показуються – вони визначаються типом таблиць. Первинний ключ таблиці факту цілком складається з первинних ключів усіх таблиць розмірності. У прикладі (таблиця факту «SALE») первинний ключ складений з чотирьох зовнішніх ключів: CustomerID, SalespeopleID, TimeID і ProductID.

Таблиці розмірності мають менша кількість рядків, чим таблиці факту і містять описову інформацію. Ці таблиці дозволяють користувачеві швидко переходити від таблиці факту до додаткової інформації.

У прикладі на рис. 3.24 , таблиця «SALE» - таблиця факту; «CUSTOMER», «TIME», «SALESPEOPLE» і «PRODUCT» - таблиці розмірності, що дозволяють витягати інформацію хто і коли зробив покупку, який продавець і на яку суму продав і які саме товари були продані.

ERwin підтримує використання вторинних таблиць розмірності, називаних консольними (outrigger) таблицями, хоча вони не потрібні для схеми зірка. Консольні таблиці можуть бути зв'язані тільки таблицями розмірності, причому консольна таблиця в цьому зв'язку батьківська, а таблиця розмірності - дочірня. Зв'язок може бути ідентифікуючої або не ідентифікуючої. Консольна таблиця не може бути зв'язана таблицею факту. Вона використовується для нормалізації даних у таблицях розмірності. Нормалізація даних корисна при моделюванні реляційної структури, але вона зменшує ефективність виконання запитів до сховища даних.

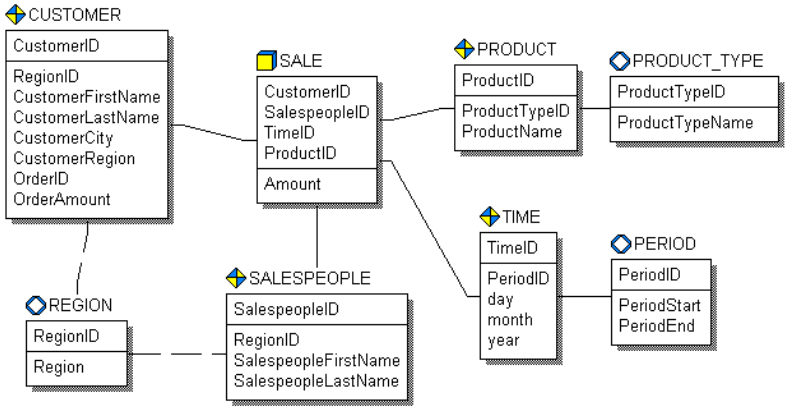


Рисунок 3.25 - Схема «сніжинка».

У розмірній моделі головною метою є забезпечення високої ефективності перегляду даних і виконання складних запитів. Схема сніжинка (рис. 3.25) звичайно перешкоджає ефективності, тому що вимагає об'єднання багатьох таблиць для побудови результуючого набору даних, що збільшує час виконання запиту. Тому при проектуванні не слід зловживати створенням безлічі консольних таблиць. Коли консольні таблиці використовуються в розмірній моделі, для нормалізації кожної таблиці розмірності, модель називається сніжинка

У діалозі опису властивостей таблиці Table Editor мається закладка Dimensional, у якій задаються специфічні властивості таблиці в розмірній моделі (рис. 3.26):

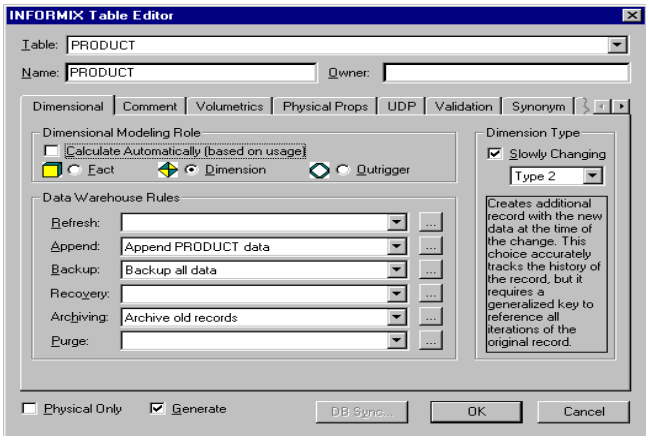


Рисунок 3.26 - Закладка Dimensional діалогу Table Editor.

Роль таблиці в схемі (Dimensional Modeling Role). За замовчуванням Erwin автоматично визначає роль таблиці на підставі створених зв'язків (таблиця факту, розмірності або консольна). Таблиця без зв'язків визначається як таблиця розмірності, таблиця факту не може бути батьківської в зв'язку, таблиця розмірності може бути батьківської стосовно таблиці факту, консольна таблиця може бути батьківської стосовно таблиці розмірності. При ручному призначенні ролі таблиці ERwin автоматично перевіряє коректність розмірної моделі і видає діалог з попереджуючим повідомленням у випадку наступних порушень синтаксису:

- Таблиця факту не є в зв'язку дочірньої;
- Консольна таблиця не є в зв'язку батьківської;
- Встановлено ідентифікуючий зв'язок між консольною таблицею і таблицею факту.

Тип таблиці розмірності (Dimension Type). Кожна таблиця розмірності може містити незмінні, або рідко змінювані дані (slowly changing dimensions). Оскільки сховище даних має ненормалізовану структуру, редагування таблиць розмірності може привести до колізій. Для того, щоб уникнути протиріч при збереженні даних, ERwin дозволяє задати тип рідко змінюваних даних, що відрізняється способом редагування даних:

- Модифікація старих даних новими, при цьому старі дані губляться.
- Створення нового запису в таблиці розмірності з новими даними і часом зміни. У цьому випадку зберігаються старі дані і можна простежити історію зміни редагування даних, але необхідно генерувати ключ для посилання на старі дані.
- Запис нових даних у додатковому полі того ж самого запису. У цьому випадку зберігається первісне й останнє нове значення. Усі проміжні дані губляться.

Правила збереження даних (Data Warehouse Rules). Для кожної таблиці можна задати шість типів правил маніпулювання даними: відновлення (Refresh), доповнення (Append), резервне копіювання (Backup), відновлення (Recovery), архивирование (Archiving) і очищення (Purge). Для завдання правила варто вибрати ім'я правила з відповідного списку вибору. Кожне правило повинне бути попередньо описане в діалозі Data Warehouse Rule Editor (меню Edit / Data Warehouse Rule). Для кожного правила повинне бути задане ім'я, тип, визначення. Наприклад, визначення правила доповнення даних може включати частоту і час доповнення (щодня, наприкінці робочого дня), тривалість операції і т.д. Зв'язати правила з визначеною таблицею можна за допомогою діалогу Table Editor.

При проектуванні сховища даних важливо визначити джерело даних (для кожного стовпчика), метод, яким вихідні дані витягаються , перетворюються, і фільтруються перш, ніж вони імпортуються в сховище даних. Сховище даних може поєднувати інформацію з текстових файлів і багатьох баз даних, як реляційних, так і нереляційних, у єдину систему підтримки прийняття рішень. Щоб підтримувати регулярні відновлення і перевірки якості даних, необхідно знати джерело для кожного стовпчика в сховище даних. Для документування інформації про джерела даних використовується редактор Data Warehouse Source Editor (рис.3.27).

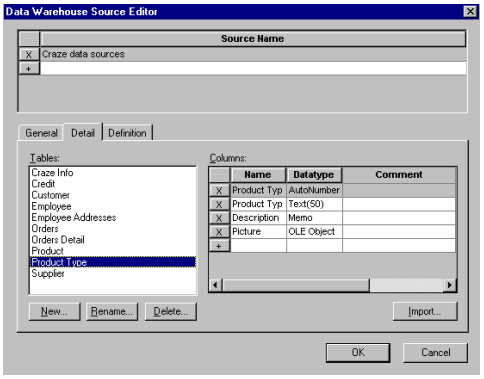


Рисунок 3.27 - Діалог Data Warehouse Source Editor.

Імена таблиць і колонок джерел даних можуть бути імпортовані як з баз даних (зворотне проектування), так і з інших моделей ERwin. Кожному джерелу може бути задані ім'я і визначення. У редакторі Column Editor необхідно додати інформацію про використання джерел даних для кожного стовпчика таблиць сховища даних, а так само додаткову інформацію про способи, режими і періодичність перенесення даних із джерела в сховище даних

Контрольні запитання

1. Етапи проектування БД, їх характеристики.
2. Основні поняття моделі “сутність-зв’язок”: сутність; ключові атрибути; зв’язок між сутностями, кардинальність.
3. Поняття наслідування сутностей.
4. Правила переходу від моделі “сутність-зв’язок” до схеми відносин реляційної моделі даних.
5. Яка різниця між сховищем и реляційної БД: недоліки, переваги.
6. Розробити проект моделі комплексного завдання.

4. Принципи нормалізації відношень

4.1 Нормальні форми і нормалізація відношень

Принцип нормалізації відношень полягає в тому, щоб наявне вихідне відношення привести до такого виду (форми), при якому усуваються небажані залежності між атрибутами, що приводять до різних аномалій при роботі з БД.

Під *аномалією* розуміють таку ситуацію, яка приводить до протиріч у БД чи до необхідності виконання яких-небудь додаткових (залишкових, непотрібних) дій, які у свою чергу приводять до невиправданого ускладнення операцій обробки даних і додаткового часу. Розрізняють аномалії модифікації, додавання і видалення даних.

Аномалії модифікації виявляються в тім, що при зміні даних виникає необхідність одночасно змінювати ще якісь дані. Наприклад, нехай мається якось відношення, що містить інформацію про співробітників. І дані організовані так, що при зміні номера телефону в кімнаті виявляється, що потрібно ще змінювати номера телефонів усіх співробітників, робочі місця яких знаходяться в цій же кімнаті. Якщо це так, це приклад аномалії, яка є наслідком небажаних залежностей між атрибутами в даному відношенні.

Аномалія видалення може виявлятися в тім, що при видаленні деяких даних може втратитися якась інша інформація, зв'язана з даними, що видаляються.

Аномалія додавання виникає у випадках, якщо дані не можна ввести доти, поки не будуть уведені ще які-небудь інші дані.

Для запобігання можливих аномалій у БД застосовується класична технологія, заснована на *теорії нормалізації відношень*. У рамках цієї теорії розроблена технологія послідовного перетворення вихідних відношень до так називаних **нормальних форм** (форми Кодда, по імені вченого, що їх запропонував). Усього розроблено 6 нормальних форм:

- перша нормальна форма (1НФ);
- друга нормальна форма (2НФ);
- третя нормальна форма (3НФ);
- нормальна форма Бойса-Кодда (БКНФ);
- четверта нормальна форма (4НФ);
- п'ята нормальна форма (5НФ).
- Домен-Ключ нормальна форма (DKNF).

Кожній нормальній формі (НФ) відповідають визначені вимоги, яким повинне задовольняти відношення, що знаходиться в даної НФ. Чим більше номер НФ, тим більш досконалою є організація даних (менше імовірність виникнення аномалій).

Загальні властивості НФ слідуючи:

- кожна наступна НФ у деякому змісті поліпшує властивості попередньої НФ;
- при переході до наступної НФ властивості попередньої НФ зберігаються.

Отже, технологія нормалізації відношень (метод нормальних форм) полягає в наступному.

На початку визначаються *вихідні відношення*. Вихідні відношення задовольняють вимозі 1НФ по визначенню.

Потім для кожного вихідного відношення перевіряється, чи задовольняє відношення вимогам 2НФ. Якщо ні, виконується декомпозиція вихідного відношення таким чином, щоб отримані в результаті декомпозиції нові відношення задовольняли вимогам 2НФ. Далі точно так само отримані відношення (чи вихідне відношення, якщо воно задовольняє вимогам 2НФ) перевіряються на їхню відповідність вимогам 3НФ. Якщо вимоги 3НФ не виконуються, виконується подальша декомпозиція відношень, і т.д. У багатьох практичних випадках досить привести усі відношення до 3НФ.

Нижче ми розглянемо докладно вимоги до 1НФ, 2НФ і 3НФ. Але тепер необхідно розглянути поняття *залежності між атрибутами відношень*, тому що вимоги НФ засновані на застосуванні цих понять.

4.2 Поняття залежності між атрибутами відношень

Існують такі типи залежностей між атрибутами: функціональна, транзитивна і багатозначна. Базовим є поняття функціональної залежності.

Атрибут **A** *функціонально залежить* від атрибута **B**, якщо кожному значенню **A** відповідає в точності одне значення атрибута **B.** Це означає, що у всіх кортежах з однаковими значеннями атрибута **A** атрибут **B** буде мати те ж саме значення.

Математично це записується так: **A → B.**

Атрибути **A** и **B** можуть бути складеними, тобто складатися з двох чи більш атрибутів.

Приклади: **ПІБ → РікНародж , Викладач → Каф.**

Атрибути **A** и **B** *функціонально взаємозалежні*, якщо існують функціональні залежності **A → B** и **B → A.**

Математично це позначається: **A ↔ B.**

Приклади: **ПІБ ↔ НомСтудКвіт , Місто ↔ ПочтІнд.**

Частковою функціональною залежністю називається залежність неключових атрибутів від частини складеного ключа відношення.

Приклад. Нехай мається відношення

УСПІШНІСТЬ (**ПІБ**, **Предм**, **РікНародж**, **Оцінка**),

у якому первинний ключ складений - включає атрибути **ПІБ** і **Предм**. Атрибут **РікНародж** знаходиться у функціональній залежності від атрибута **ПІБ,** що є частиною складеного ключа. Отже, атрибут **РікНародж** знаходиться в частковій функціональній залежності від первинного ключа відношення.

Повна функціональна залежність має місце, якщо неключовий атрибут функціонально залежить від усього складеного ключа.

У попередньому прикладі атрибут **Оцінка** знаходиться в повній функціональній залежності від первинного ключа.

Атрибут **C** знаходиться в *транзитивній залежності* від атрибута **A**, якщо існує атрибут **B** такий, що **A → B** и **B → C.**

Приклад: **ПІБ → Місто → ПоштІндекс**.

Між атрибутами також можуть існувати багатозначні залежності.

У відношенні *R* атрибут **B** знаходиться в *багатозначній залежності* від атрибута **A**, якщо кожному значенню атрибута **A** відповідає множина значень атрибута **B**, не зв'язаного залежністю з іншими атрибутами з *R.*

Приклад. Нехай є відношення

ВИКЛАДАЧ (**ПІБ**, **Посада**, **Оклад**, **Предм**, **Група**).

Між атрибутами **ПІБ** і **Предм** існує багатозначна залежність: один викладач може вести заняття по декількох предметах, і навпаки, один предмет можуть вести декілька викладачів.

Багатозначні залежності можуть бути наступних типів: “один до кількох” (1:M), “кілька до одного” (M:1) і “ кілька до кількох” (M:M).

При проектуванні БД завжди прагнуть перетворити відношення з багатозначними залежностями атрибутів типу M:M в відношення з залежностями типу 1:M чи M:1. Одержувані при цьому нові відношення виявляються зв'язаними типом зв'язку, який відповідає типу залежності між атрибутами.

Приведене вище відношення ВИКЛАДАЧ можна перетворити в двоє відносин ВИКЛАДАЧ і УЧЕБН_НАВАНТАЖЕННЯ , наприклад, таким чином:

ВИКЛАДАЧ (ПІБ, Посада, Оклад) і

УЧЕБН_НАВАНТАЖЕННЯ (ПІБ, Предм, Група).

Перше відношення з другим зв'язується зв'язком типу 1:M по атрибуту ПІБ.

Атрибути можуть бути *функціонально незалежними*. Два і більш атрибути є взаємно незалежними, якщо жоден з них не знаходиться у функціональній залежності від іншого атрибута.

Основний спосіб виявлення залежності між атрибутами – це уважний аналіз семантики (змісту) атрибутів.

Розглянемо наступний приклад. Нехай є відношення ВИКЛАДАЧ, у якому представлені такі атрибути:

* КодВикл - код викладача;
ПІБ - прізвище, ім'я і по батькові;
Каф - кафедра;
Посада - посада;
Оклад - оклад;
* Предм - предмет;
ВидЗанят - вид заняття.

Потрібно виявити наявні функціональні залежності між цими атрибутами.

На початку необхідно визначити первинний ключ відношення. Неважко бачити, що в даному прикладі первинний ключ складений і включає два атрибути: **КодВикл** і **Предм**. Ці атрибути однозначно ідентифікують кожен кортеж відношення. Між атрибутами, що ввійшли в первинний ключ, залежності не встановлюються.

У результаті зробленого аналізу виявлені функціональні залежності між атрибутами такі, що у виді схеми зображені на рис. 4.1.

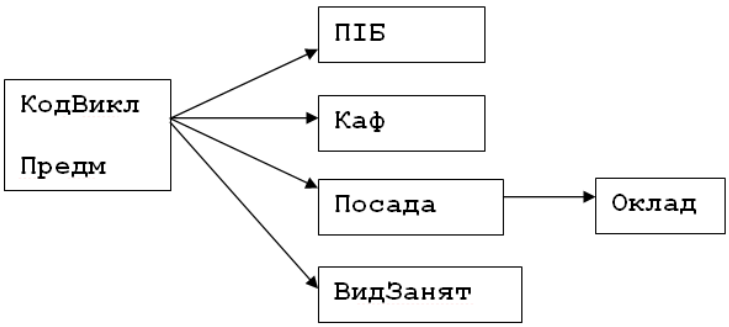


Рисунок 4.1 - Схема залежності між атрибутами вихідного відношення ВИКЛАДАЧ

Обґрунтування виявленої залежності може бути наступне.

Прізвище, кафедра і посада завжди однозначно визначені для кожного викладача. Тому атрибути **ПІБ**, **Каф** і **Посада** функціонально залежать від атрибута **КодВикл**.

Оклад викладача однозначно визначається його посадою (для спрощення ми тут не враховуємо фактори стажу, ученого ступеня, і інші).

Отже, існує функціональна залежність атрибута **Оклад** від атрибута **Посада**.

Один викладач по одному предметі може проводити різні види занять. Тому атрибут **ВидЗанят** повинний знаходитися в повній функціональній залежності від атрибутів **КодВикл** і **Предм**, тобто від первинного ключа відношення.

Виявлені залежності між атрибутами в даному прикладі відбивають об'єктивно існуючі залежності між властивостями сутності. Питання про те, бажані вони чи небажані, як позбутися від небажаних залежності розглянемо далі.

4.3 Перетворення відношень до 1, 2 і 3 нормальних форм

Тепер розглянемо конкретні вимоги, яким повинні задовольняти відношення у 1НФ, 2НФ і 3НФ, і проілюструємо на прикладах правила перетворення (декомпозиції) відносин для приведення їхній до тієї чи інший НФ.

Перша нормальна форма (1НФ). Відношення знаходиться в 1НФ, якщо всі його атрибути прості (неподільні).

Вихідне відношення завжди знаходиться в 1НФ, тому що по визначенню відношення його атрибути повинні бути простими. На рис.4.2 наведено етапи звореня таблиць у перщої НФ.

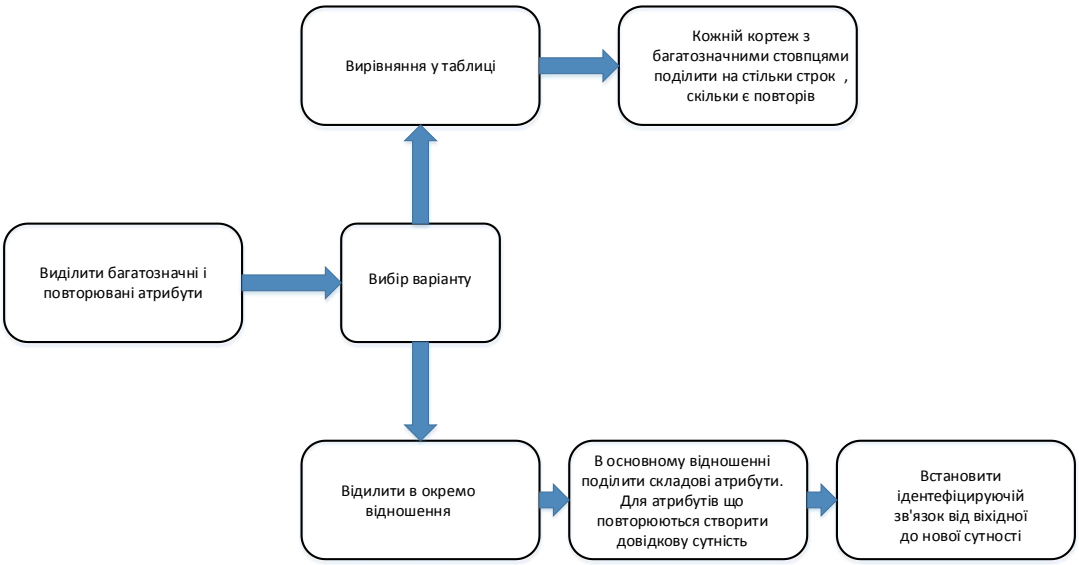


Рисунок 4.2 - Етапи створення таблиць у першої НФ

Приклад.

Таблиця Співробітники має повторюючи атрибути Спеціальність та Телефон. Поле Прізвище , ім'я та по-батькові та Дата зачислення та звільнення багато значні. Для створення таблиць у першої НФ виконано розподіл атрибуту та створено довідкові таблиці (рис.4.3).

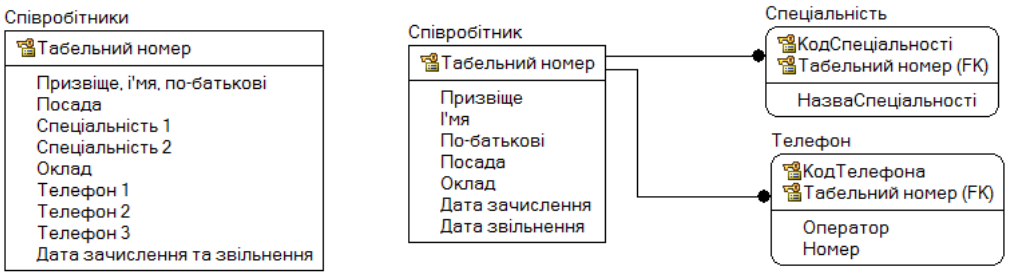


Рисунок 4.3 Перехід до першої НФ

Друга нормальна форма (2НФ). Відношення знаходиться в 2НФ, якщо воно знаходиться в 1НФ і кожний його неключовий атрибут функціонально повно залежить від первинного ключа.

У розглянутому вище прикладі відношення ВИКЛАДАЧ не задовольняє вимогам 2НФ. Неключові атрибути **ПІБ**, **Каф** і **Посада** знаходяться в частковій функціональній залежності від первинного ключа.

Теорема Хиса.

Як що у відношенні з схемою **R(S)**, де **S** – повний набір атрибутів відношення (рис.4.4), створені три набору атрибутів **A,B,C**, таких що **A ∩ B ≠0**, **A ∩ C ≠0**, **B ∩ C ≠0**, **A ∩ B ∩ C = S** то, як що набор **C** функціонально залежіть від **B,** то проекції побудують повну декомпозицію відношення **r.**

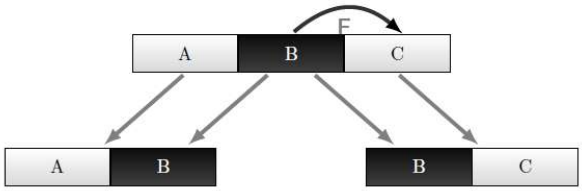


Рисунок 4.4 - Пояснення теореми Хиса

Для перетворення відношення до 2НФ виконується його декомпозиція за наступними правилами (рис .4.5)

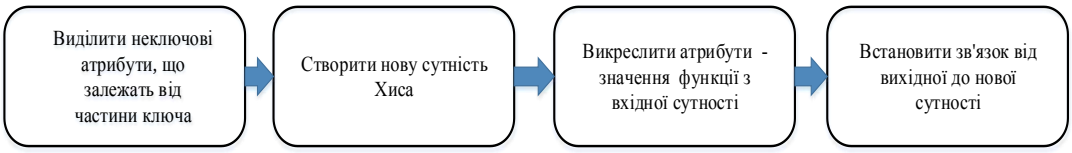


Рисунок 4.5 –Перетворення відношення до 2НФ

Приклад 1.

А) Побудувати проекції відношення на атрибути, які знаходяться в повній функціональній залежності від первинного ключа.

Для відношення ВИКЛАДАЧ така проекція єдина. Вона включає атрибути: **КодВикл**, **Предм**, **ВидЗанят**. Одержимо нове відношення, що доцільно назвати ЗАНЯТТЯ. Схема залежності між атрибутами приведена на рис. 4.6.

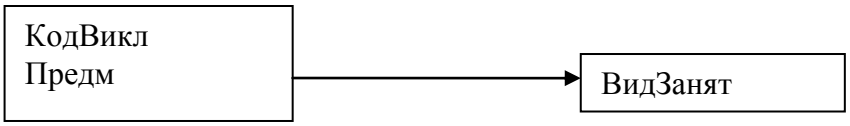


Рисунок 4.6 - Схема залежності між атрибутами відносини ЗАНЯТТЯ

Б) Побудувати проекції на частині складеного первинного ключа й атрибути, що знаходяться у функціональній залежності від цих частин.

У розглянутому прикладі тільки одна така частина – атрибут **КодВикл**. Одержувана проекція є відношенням, яку назвемо ВИКЛАДАЧ. Схема залежності атрибутів отриманого відношення показана на рис. 4.7.

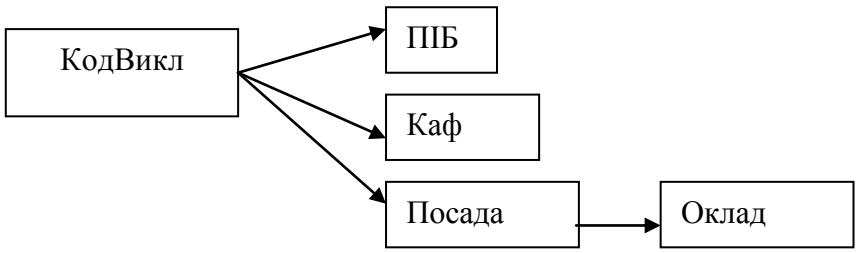


Рисунок 4.7 - Схема залежностей між атрибутами відношення ВИКЛАДАЧ,

Отже, у результаті зробленої декомпозиції вихідного відношення ми одержали два нових відношення: ЗАНЯТТЯ і ВИКЛАДАЧ. Неважко бачити, що отримані відношення задовольняють вимогам 2НФ.

Приклад 2. Перетворення сутності «Проект» до другої НФ (рис. 4.8).

Рисунок 4.8 -Перетворення сутності до другої НФ

Третя нормальна форма (3НФ). Відношення знаходиться в 3НФ, якщо воно знаходиться в 2НФ і кожний його неключовий атрибут нетранзитивно залежить від первинного ключа. У розглянутому прикладі відношення ЗАНЯТТЯ знаходиться в 3НФ. Відношення ВИКЛАДАЧ не задовольняє вимогам 3НФ, тому що атрибут **Оклад** від первинного ключа залежить транзитивно.

Транзитивна залежність є небажаної, тому що приводить до надлишкового дублювання інформації. Для усунення транзитивної залежності необхідно побудувати проекції на атрибути, що беруть участь у транзитивній залежності.

Проекція на атрибути **Посада** і **Оклад** дає відношення, що назвемо ОКЛАДИ. Схема залежності між атрибутами зображена на рис. 4.9.

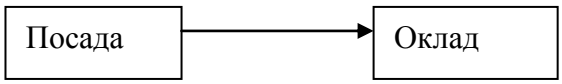


Рисунок 4.9 - Схема залежностей між атрибутами відношення ОКЛАДИ

Проекція на атрибути **КодВикл**, **ПІБ**, **Каф** і **Посада** дасть відношення ВИКЛАДАЧ. Схема залежності між його атрибутами показана на рис. 4.10.

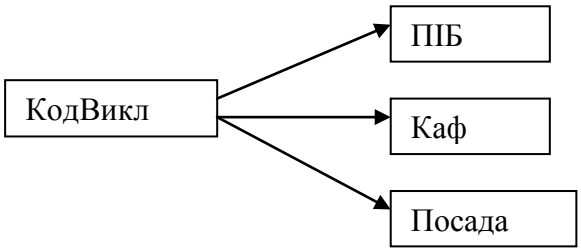


Рисунок 4.10 - Схема залежностей між атрибутами відношення ВИКЛАДАЧ, перетвореного до 3НФ

Отже, у результаті послідовної декомпозиції вихідного відношення ВИКЛАДАЧ (Рис. 4.1) ми одержали наступні три відношення:

ЗАНЯТТЯ (**КодВикл**, **Предмет**, **ВидЗанят**);

ОКЛАДИ (**Посада**, **Оклад**);

ВИКЛАДАЧ (**КодВикл**, **ПІБ**, **Каф**, **Посада**).

Кожне з цих відношень задовольняє вимогам 1, 2 і 3 НФ. У більшості практичних випадків цього досить для виключення аномалій і нормальної роботи БД. Отримані відношення повинні бути зв'язані між собою. Схема зв'язків зображена на рис. 4.11.

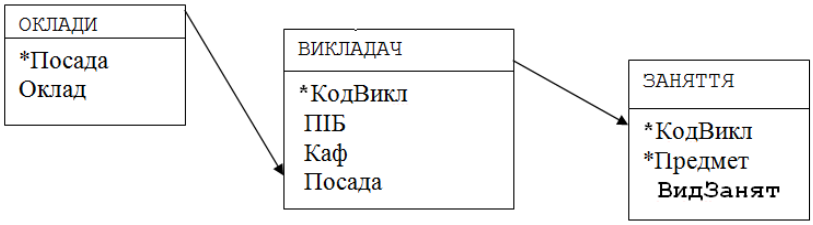


Рис. 4.11 Схема зв'язків між відношеннями, отриманими в результаті нормалізації вихідного відношення ВИКЛАДАЧ

4.4 Файлова організація даних

У різних СУБД збереження і доступ до даних організовані по різному. Однак, існують деякі загальні принципи, що застосовуються практично у всіх СУБД. Розглянемо їх основні положення. Дані завжди зберігаються у файлах.

Файлом називається пойменована лінійна послідовність записів, розташованих на зовнішніх носіях.

Записом звичайно називають сукупність даних, що представляють інформацію одного кортежу з деякого відношення (одного рядка таблиці).

Зовнішні носії інформації підрозділяються на пристрої з довільною адресацією (магнітні й оптичні диски) і пристрої з послідовною адресацією (магнітофони, стрімери). На пристроях з довільною адресацією установка голівок запису-читання на потрібну адресу виконується практично миттєво. Час позиціювання голівок дуже малий в порівнянні з часом зчитування запису. Такі пристрої називають також **пристроями прямого доступу.**

У пристроях з послідовною адресацією для установки голівок на потрібну адресу потрібно послідовно "переглянути" усі попередні записи. Такі пристрої називають **пристроями послідовного доступу**. Для збереження БД такі пристрої не застосовуються.

Файли, розміщені на пристроях прямого доступу, які утримують записи постійної довжини, називають **файлами прямого доступу**. У таких файлах до будь-якого запису можна звертатися по фізичній адресі розташування запису на диску. Якщо записи у файлі мають довільну довжину, то звертатися до потрібного запису по його адресі не можна – потрібно послідовно переглянути всі попередні записи. Такі файли називають **файлами послідовного доступу**, незважаючи на те, що вони можуть розміщатися на пристрої прямого доступу.

Однак, на жаль, не завжди можна зберігати інформацію у вигляді файлів прямого доступу. Тому для БД винятково важливою є проблема швидкого доступу до даних. Причому суттю проблеми є навіть не стільки неможливість прямого доступу до фізичного запису, а насамперед взагалі низька ефективність доступу до записів по їхній фізичній адресі.

Розроблено дуже велика кількість засобів для подолання цієї проблеми. Однак основним засобом забезпечення швидкого доступу до даних є індексування.

4.5. Індексування

Під **індексом** у загальному випадку розуміється засіб (технологія) прискореного доступу до даних. **Фізично** індекс являє собою область пам'яті чи окремий файл (індексний файл), у якому в упорядкованій послідовності розміщені всі значення поля, для якого створений цей індекс (точніше, згортки значень полів, які мають фіксований розмір), і з кожним значенням поля зберігається також посилання на запис (адреса) , у якій знаходиться значення цього поля.

Пошук запису, у якій міститься задане значення поля (ключа), виконується по індексі в такий спосіб. На початку виконується пошук значення поля в індексі. Завдяки упорядкованості значень поля в індексі пошук виробляється дуже швидко тому, що застосовується так званий "бінарний пошук" (іноді "логарифмічний пошук"), заснований на відомому *методі розподілу навпіл*. Зі знайденої в індексі запису визначається адреса основного запису, у якому міститься шукане значення. По цій адресі і виконується звертання до потрібного запису.

Крім того, висока швидкість досягається завдяки тому, записи в індексному файлі завжди мають постійну довжину.

Така загальна схема застосування індексу. Відомі різні варіанти реалізації цієї загальної схеми. Розглянемо найбільш розповсюджені з них.

4.5.1 Файли з щільним індексом

В основній області файлу розміщається послідовність записів однакової довжини, розташованих у довільному порядку. В індексній області розміщені індексні записи так само постійної довжини, які мають таку структуру:

Значення ключа (чи його згортка)	Номер запису

Тут значення ключа – це значення поля (це може бути первинний чи зовнішній ключ, або звичайне поле), для якого створюється даний індекс. Замість значення ключа звичайно зберігають його згортку (чи хеш-код), що має завжди невелику і фіксовану довжину, наприклад, 4 байти. Усі записи в індексній області упорядковані за значеннями ключа. Завдяки цьому в індекс-

ній області можна застосувати ефективний бінарний пошук, що вимагає мінімальних витрат часу на пошук.

Ефективність того чи іншого методу пошуку прийнято оцінювати максимальним часом доступу до довільного запису. Прийнято **час доступу** визначати не в абсолютних значеннях часу, а в кількості звертань до пристрою зовнішньої пам'яті, яким звичайно є диск. У випадку бінарного пошуку максимальна кількість звертань до диску (кроків пошуку) дорівнює:

$T_{\text{пошуку}} = \log_2 N$,

де N – число записів, серед яких виконується пошук. Тому бінарний пошук часто називають також логарифмічним пошуком.

На диску записи зберігаються в блоках. Розмір блоку визначається фізичними параметрами дискового контролера і настроюваннями операційної системи. В одному блоці можуть розміщатися кілька записів.

На рис. 4.12 показаний приклад організації файлу з щільним індексом.

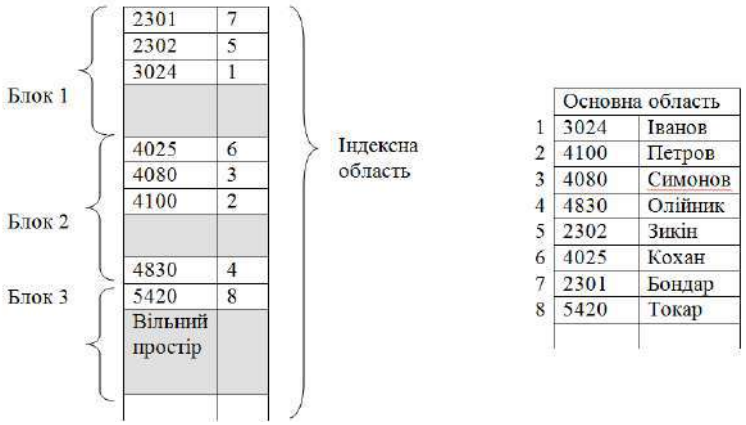


Рисунок 4.12 - Приклад організації файлу з щільним індексом

Для оцінки виграшу, що дає використання щільного індексу, розглянемо такий приклад. Нехай відомі такі характеристики:

- розмір блоку LB = 1024 байта;
- довжина запису у файлі LZ = 256 байтів;
- довжина первинного ключа LK = 4 байти;
- кількість записів у файлі KZ = 100000.

Розрахуємо розмір індексного запису LI у такий спосіб. Для збереження номера запису нам буде потрібно 3 байти:

$2^8 = 256$ - 1 байт;

$2^{16} = 65536$ - 2 байти;

$2^{24} = 16777216$ - 3 байти.

Число 100000 можна зберігати у області пам’яті розміром не менше трьох байтів. Звичайно допускається тільки парна адресація. Тому для збереження номера запису відводимо 4 байти. З обліком цього довжина індексного запису буде дорівнювати:

$LI = LK + 4 = 4 + 4 = 8$ байт.

Число індексних записів, що можуть зберігається в одному блоці, дорівнює:

$KIZB = 1024/8 = 128$.

Тепер знайдемо необхідну кількість індексних блоків:

$KIB = 100000/128 = 781{,}25 \approx 782$ блоку.

Тут результати ми округляємо у більшу сторону, тому що пам'ять виділяється тільки цілими блоками.

Обчислимо максимальну кількість звертань до диска при пошуку довільного запису в такий спосіб:

$T_{\text{пошуку}} = \text{Log}_2 KIB + 1 = \text{Log}_2 728 + 1 \approx 10 + 1 = 11$.

Тут ми так само округляємо результат до найближчого цілого. Одиниця додана для обліку звертання до запису в основній області.

Отже, ми визначили, що для пошуку довільного запису в даному прикладі при застосуванні щільного індексу буде потрібно не більш 11 звертань до диска.

Тепер визначимо, скільки нам треба було б звертань до диска у випадку, якби індекс не створювався. Для цього треба було б у гіршому випадку переглянути всі блоки, у яких зберігаються записи в основній області.

Часом пошуку усередині блоку ми зневажаємо, тому що цей процес протікає в оперативній пам'яті.

Кількість блоків основної області, необхідних для збереження всіх 100000 записів, дорівнює:

$KBO = KZ/(LB/LZ) = 100000/ (1024/256) = 100000/4 = 25000$ блоків.

Це значить, що максимальна довжина доступу при безпосередньому пошуку довільного запису дорівнює 25000 звертань до диска. Раніше ми визначили, що с застосуванням індексу достатньо лише 11 звертань. Як бачимо, виграш дуже істотний.

Тепер розглянемо, як будуть здійснюватися операції додавання і видалення записів при застосуванні щільного індексу.

При операції додавання запис міститься в кінець основної області. При цьому в індексну область потрібно помістити відповідний індексний запис таким чином, щоб зберігалася упорядкованість ключових значень в індексі. Для цього при створенні індексної області в кожнім блоці залишається вільний простір (так називаний відсоток розширення).

Після знаходження блоку, у який потрібно помістити індексний запис, цей блок копіюється в оперативну пам'ять, там новий індексний запис уставляється на потрібне місце і після цього змінений блок знов записується на диск на колишнє місце.

Таким чином, максимальна кількість звертань до диска, що потрібно при додаванні нового запису – це кількість звертань при пошуку потрібного індексного блоку плюс 1 звертання для занесення зміненого індексного блоку і плюс 1 звертання для занесення запису в основну область: $T_{додав} = \mathrm{Log}_2 KIB+1+1 = \mathrm{Log}_2 728+1+1 \approx 10+1+1 = 12$.

Природно, у міру додавання нових записів відсоток розширення (вільний простір) буде зменшуватися. Коли зникне вільний простір, відбувається переповнення індексної області. У цьому випадку можливі два варіанти рішення: або перешикувати індексну область заново, або організувати область переповнення для індексної області, у якій розміщати записи, яки не помістилися в індексній області. У першому випадку буде потрібно витрачати додатковий час на перебудову індексної області. В другому випадку збільшиться час доступу до довільного запису і буде потрібно зберігати в індексному запису додаткові посилання на область переповнення. Саме тому при проектуванні БД важливо заздалегідь по можливості точніше визначити обсяги інформації, яка буде зберігатися, спрогнозувати ріст обсягу інформації і на основі цього розрахувати необхідний обсяг відсотка розширення в індексних блоках.

Видалення записів виконується в такий спосіб. Запис в основній області позначається як вилучений (відсутній). В індексній області відповідний індексний запис знищується фізично, тобто всі наступні за ним записи переміщаються таким чином, щоб заповнити місце вилученого попереднього запису. Ці дії виконуються в оперативній пам'яті. Таким чином, кількість звертань до диска при видаленні запису таке ж, як і при додаванні нового запису.

Доступ до даних, заснований на використанні щільного індексу називають **індексно-прямим доступом**, а файли з щільним індексом називають часто індексно-прямими файлами.

4.5.2. Файли з нещільним індексом чи індексно-послідовні файли

Нещільний індекс використовується у файлах, у яких записи в основній області зберігаються в упорядкованому виді. Записи в індексній області мають наступну структуру:

Значення ключа (згортки) першого запису блоку	Номер блоку, у якому зберігається запис

В індексній області записи містять ключі (згортки) перших записів із блоків основної області. На рис. 4.13 показаний приклад заповнення індексної й основної області при нещільному індексі.

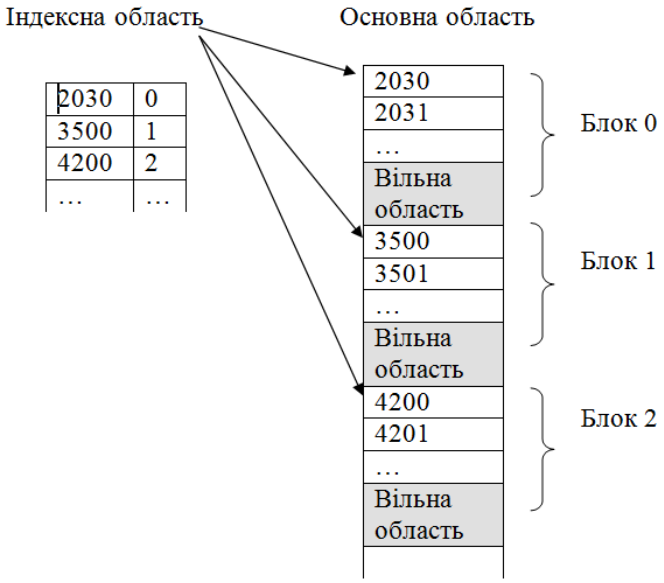


Рисунок 4.13 - Приклад індексної й основної області при нещільному індексі

В індексній і основній областях записи упорядковані. Пошук запису виконується наступним чином. В індексній області за заданим значенням ключа виконується пошук номера блоку, у якому знаходиться потрібний запис. Тому що всі записи в основній області упорядковані, знайдений блок буде завжди містити шуканий запис. Потім виконується зчитування блоку в оперативну пам'ять і там уже відбувається пошук і обробка потрібного запису. Після цього блок перезаписується на колишнє місце.

Звичайно, сортування даних в основній області вимагає значних витрат часу. Але оскільки один раз відсортовані файли при їхньому створенні потім тільки підтримуються у відсортованому виді, накладні витрати в процесі додавання нової інформації будуть значно менше.

Оцінимо довжину доступу до довільного запису для файлу з нещільним індексом. Як і раніше будемо думати, що у файлі зберігаються 100000 записів. Раніше ми уже визначили, що для їхнього збереження в основній області потрібно 25000 блоків. Для посилань на таку кількість блоків досить в індексному запису відвести 2 байти. Тоді довжина індексного запису дорівнює:

$LI = LK + 2 = 4 + 2 = 6$ байт.

Кількість індексних записів в одному блоці буде дорівнювати:

$KIZB = LB/LI = 1024/6 = 170{,}6 \approx 171$ запис.

Визначимо кількість індексних блоків, яка необхідна для збереження індексних записів:

$KIB = KBO/KIZB = 25000/171 = 146{,}2 \approx 147$ блоків.

Тепер визначимо довжину доступу при звертанні до довільного запису:

$T_{\text{пошуку}} = \text{Log}_2 KIB + 1 = \text{Log}_2 147 + 1 \approx 8 + 1 = 9.$

Отже, ми бачимо, що у випадку нещільного індексу час доступу скоротилося до 9 у порівнянні з 11 при щільному індексі.

Додавання нового запису у випадку нещільного індексу виробляється так. Спочатку шукається необхідний блок в основній області, у якому потрібно помістити новий запис. Знайдений блок основної області зчитується в оперативну пам'ять і там дописується в нього новий запис. Змінений блок потім зберігається на колишнім місці. При додаванні нового запису індексна область не корегується.

Тут так само задається відсоток розширення, але вже стосовно до основної області.

Видалення запису виробляється шляхом її фізичного видалення з основної області. Корегування індексної області при цьому так само не потрібно.

4.5.3. Організація індексів у виді В-дерев

Ідея побудови В-дерев проста і полягає в наступному. Спочатку будується нещільний індекс, як це було описано вище. Потім індексна область розглядається як основний файл і для нього ще будується нещільний індекс. Потім знову над новим індексом будується індекс наступного рівня і так доти, поки не виявиться, що індексна область складається з одного блоку. У результаті одержуємо деяке дерево, у якому кожен батьківський блок зв'язаний з однаковою кількістю підлеглих блоків, число яких дорівнює числу індексних записів, розміщених в одному блоці. Кількість звертань до диска для пошуку довільного запису при використанні такого багаторівневого індексу буде завжди однаково (для будь-якого запису) і дорівнювати кількості рівнів в отриманому дереві. Такі дерева називають *збалансованими* (від англ.. *balanced*), звідси назва В-дерево. На рис. 4.14 зображена схема, що пояснює організацію індексу у виді В-дерева.

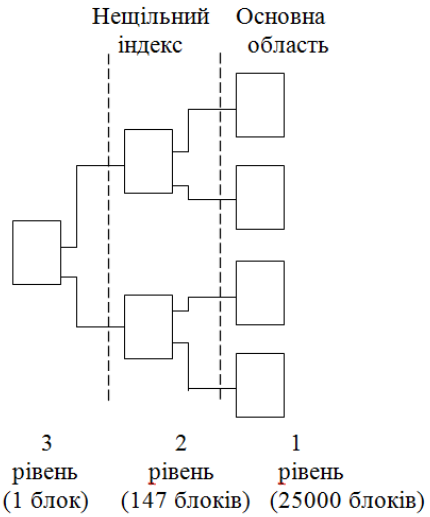


Рисунок 4.14 - Приклад організації індексу у вигляді В-дерева

Побудуємо В-дерево для умов нашого прикладу. На першому рівні число блоків дорівнює числу блоків основної області, тобто дорівнює 25000 блоків. Другий рівень утвориться як нещільний індекс над основною областю. Його ми вже будували, кількість блоків у ньому – 147. Тепер над цим другим рівнем знову побудуємо нещільний індекс.

Довжину індексного запису будемо вважати колишньою і рівною 6 байтам. Кількість індексних записів в одному блоці ми також уже визначали, вона дорівнює - 171 запис. Визначимо, скільки потрібно блоків, що необхідні для індексної області наступного 3-го рівня. Оскільки індексна область попереднього рівня містить 147 блоків, а в одному блоці можна розмістити посилання на 171 блок, то ясно, що на 3 рівні досить одного блоку.

Отже, ми одержали для розглянутого прикладу дерево, що складається з 3 рівнів (Рис. 4.9). Це значить, що для доступу до довільного запису у випадку індексу у виді В-дерева буде необхідно лише 3 звертання до диска. Причому це не максимальна кількість звертань, а завжди та сама, однакова для доступу до будь-якого запису.

Механізм додавання і видалення записів при організації індексу у виді В-дерева аналогічний розглянутому вище механізму для нещільного індексу.

4.6 Первинні і вторинні індекси

Для первинного ключа індекс створюється автоматично практично у всіх СУБД. Але при роботі з БД доводиться робити пошук по будь-якому полю, не тільки по первинному ключу. Тому у всіх СУБД передбачається можливість створення **вторинних** (чи користувальницьких) індексів, що можуть створюватися для не ключових полів. Оскільки створення кожного нового індексу завжди вимагає додаткових витрат дискової пам'яті, рекомендується вторинні індекси створювати тільки для тих полів, по яких передбачається найчастіше робити пошук даних.

Створення вторинних індексів, як правило, засноване на використанні первинного індексу. В індексній області вторинного індексу містяться у відсортованому виді хеш-коди (згортки) значень індексованого поля з відповідними посиланнями на первинний індекс. При пошуку по вторинному індексі спочатку визначається відповідний потрібному запису первинний ключ, а потім через цей ключ і первинний індекс здійснюється доступ до потрібного запису в основній області файлу. На рис. 4.15 приведена узагальнена схема, що пояснює механізм використання вторинних індексів.

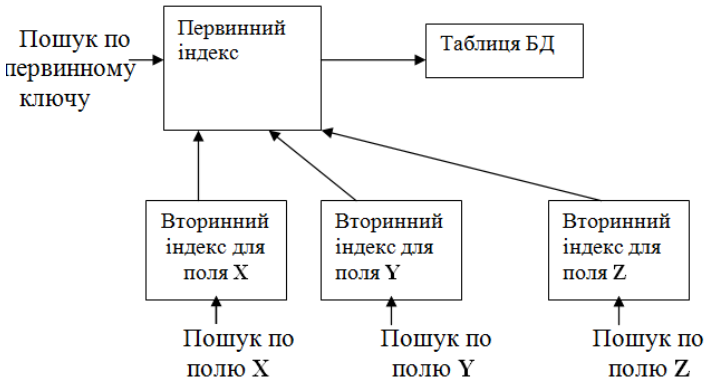


Рисунок 4.15 - Схема використання вторинних індексів

У деяких СУБД, наприклад, у Access, розподіл індексів на первинні і вторинні не виконується. Вторинні індекси можуть знищуватися, чи змінюватися, створюватися заново.

Контрольні запитання

1. Для чого потрібна нормалізація відношень?
2. Загальні властивості нормальних форм.
3. Поясніть поняття функціонально-повної, часткової і транзитивної залежностей між атрибутами відношень. Наведіть приклади.
4. Вимоги до 1 НФ і 2 НФ.
5. Правило перетворення відношень в 2 НФ.
6. Вимоги до 3 НФ. Правило перетворення відношень в 3 НФ.
7. Яка різниця між файлами прямого та послідовного доступу: недоліки, переваги.
8. Ідея та механізм роботи індексу. Первинні та вторинні індекси.
9. Файли з щільним індексом. Ідея та механізм роботи.
10. Файли з нещільним індексом. Ідея та механізм роботи.
11. Організація індексу у виді B-дерева. Ідея та механізм роботи, переваги.

5. СУБД *MS Access*. Створення таблиць баз даних

5.1. Загальні відомості про СУБД MS Access

СУБД *MS Access* є системою керування реляційними БД і включає всі необхідні інструментальні засоби для створення прикладних програм БД. MS Access дозволяє створювати: локальну БД, загальну БД у локальній мережі з файловим сервером, програму користувача для роботи з БД на *SQL*-сервері.

Останньою версією є *MS Access 2016*, що відрізняється від попередньої версії (*MS Access 2013* головним чином розвитком таких двох технологічних напрямків:

- Підтримка імпорту або зв'язування даних з Salesforce. Тепер можна імпортувати дані з Salesforce, хмарного бізнес-рішення для управління ресурсами клієнта (CRM), або створити зв'язок з цими даними.
- Підтримка імпорту або зв'язування даних з Dynamics 365. Тепер ми додали мож-ливість зв'язування і імпорту даних з Dynamics 365, хмарного бізнес-рішення для управління ресурсами клієнта (CRM).Далі для скорочення будемо використовувати просто найменування *Access*.

У *Access* реалізована об'єктно-орієнтована технологія програмування. Основними об'єктами в *Access* є наступні: таблиці, запити, форми, звіти, макроси, модулі.

Таблиця – це об'єкт, призначений для збереження даних.

Запит – об'єкт, що дозволяє користувачу одержувати потрібні дані з однієї чи декількох таблиць. За допомогою запитів можна також обновляти, додавати нові чи видаляти існуючи дані.

Форма – це об'єкт, що дозволяє створювати діалоговий інтерфейс програми користувача. Форма найчастіше створюється для зручного введення даних, відображення їх на екрані чи для керування роботою програми користувача.

Звіт – об'єкт, призначений для формування вихідних документів, що містять результати рішення задач користувача, і також виводу результатів на друкування.

Макрос – об'єкт, що містить програму, яка виконується при настанні деякої події. Макрос складається з макрокоманд. Кожна макрокоманда виконує визначені дії.

Модуль – об'єкт, що містить код програми на мові *VBA* (*Visual Basic* для прикладних програм);

5.2. Запуск Access. Створення нової БД

Запуск програми *Access* виконується точно так само, як і запуск будь-якої прикладної програми *Windows*. Наприклад, це можна зробити через головне меню: **Пуск**│**Все програми**│***Microsoft Access***. При запуску *Access* на екрані з'являється діалогове вікно, у якому потрібно вибрати один із наступних варіантів продовження роботи:

- **Нова база даних** - створюється нова база даних;
- **Запуск майстра** - майстер надає можливість скористатися наявними шаблонами (заготівками) БД (пропонуються 22 шаблона);
- **Відкрити БД** - надається можливість вибрати і відкрити одну з вже існуючих БД.

Вікно має вид, показаний на рис. 5.1.

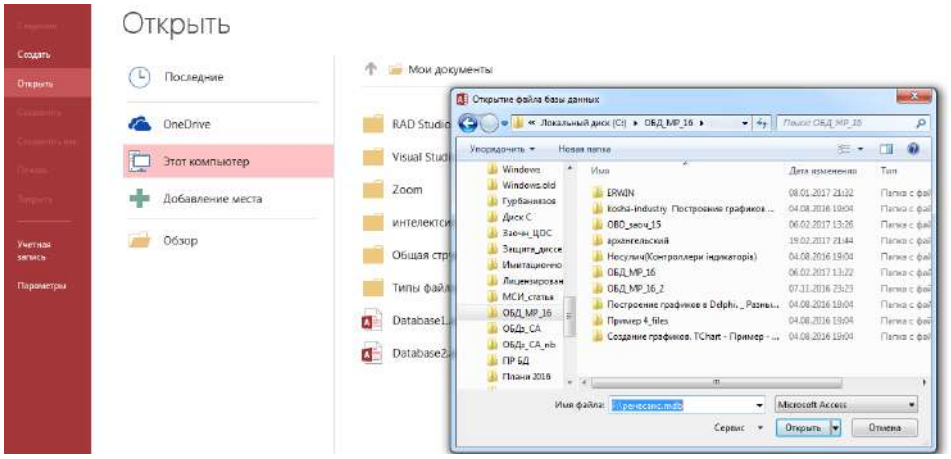


Рисунок 5.1 - Вікно вибору наступних дій в *Access*

Для створення нової бази даних потрібно вибрати варіант **Пуста база даних** (щигликом миші) і потім натиснути кнопку **ОК** (рис.5.2).

Рисунок 5.2 - Вікно вибору шаблона БД

Відкриється діалогове вікно “Файл нової бази даних”, у якому потрібно ввести ім'я нової БД, яка створюється, і вказати місце (папку), де віна буде розміщуватися (рис.5.3).

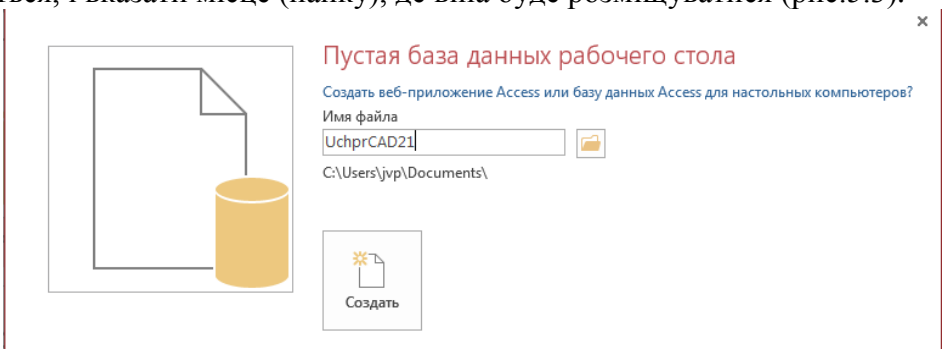


Рисунок 5.3. Вікно реєстрації БД

Ім'я файлу БД може містити до 255 символів (в імені заборонені такі символи: \ ? : * ″ < > |).

Файл БД *Access* має розширення **.accdb**. Вводити розширення не обов’язково. Якщо ви не введете розширення, *Access* автоматично додає його до імені БД.

Після уведення імені БД натисніть кнопку **Створити**. Буде створена “порожня” БД, на екрані з'явиться вікно БД (рис. 5.4). Для створення якогось об’єкту потрібно вибрати відповідну сторінку і натиснути кнопку **Створити**.

Для початкового ознайомлення з можливостями СУБД *Access* достатньо розглянути технології створення тільки найбільш важливих об’єктів: таблиць, запитів, форм.

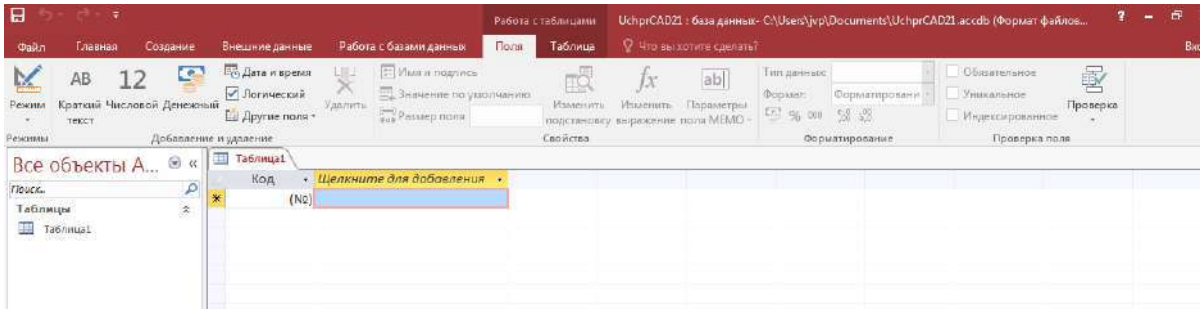


Рисунок 5.4 - Вікно “порожньої” БД

5.3 Створення таблиць БД у Access

Коли говорять про створення таблиць *Access*, мають на увазі два етапи:

- на першому етапі створюється *структура таблиці*. При цьому визначаються: склад полів, їхні імена, послідовність розміщення полів у таблиці, тип даних кожного поля, властивості полів, ключі, індекси. Всі ці елементи об'єднується поняттям “структура таблиці”. На цьому етапі даних у таблиці ще немає;
- на другому етапі виконується заповнення таблиці даними.

Ми поки розглянемо тільки питання створення структури таблиці.

У *Access* передбачено кілька способів (режимів) створення таблиць, основним з яких варто вважати створення таблиці за допомогою Конструктора. Для відкриття Конструктора таблиць потрібно виконати наступне:

- у вікні БД ліворуч у панелі «Все Об’єкти» відкрити вкладку ”Таблиці” і відкрити закладку **Створити** та натиснути кнопку **Конструктор Таблиць**;
- відкриється вікно “Нова таблиця”, у якому потрібно вибрати режим “Конструктор” і натиснути кнопку **ОК** (рис. 5.5).

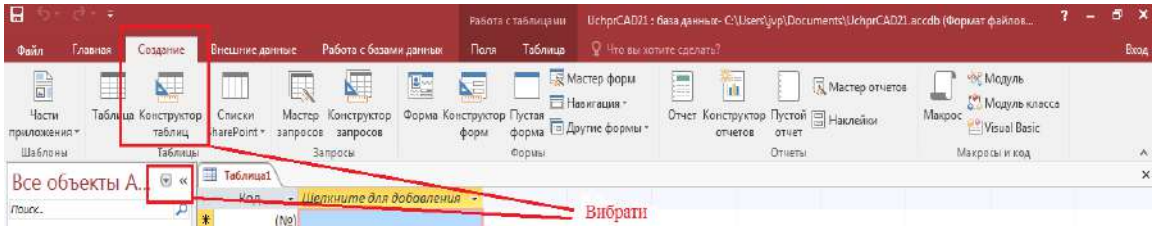


Рисунок 5.5. Вікно нова таблиця

Далі в вікні Конструктора таблиць (рис.5.6) будуть виконуватися всі основні дії по створенню структури таблиці.

Процес створення таблиці за допомогою Конструктора включає такі етапи:

- визначення полів таблиці;
- створення первинного ключа;
- збереження створеної структури таблиці в БД.

Розглянемо кожний з цих етапів більш докладно.

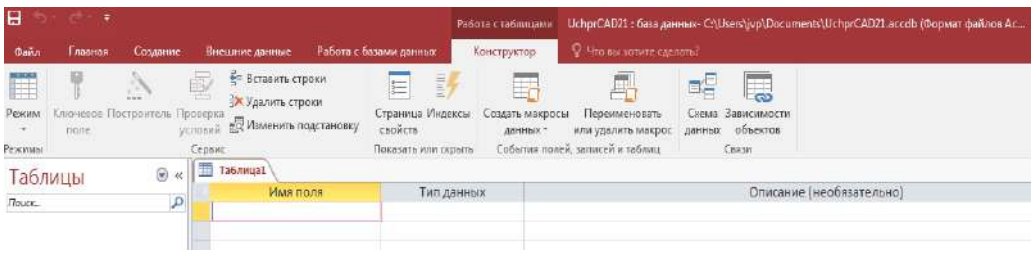


Рисунок 5.6. Вікно Конструктора таблиць

5.3.1 Визначення полів таблиці

Визначення полів таблиці включає створення:

- імен полів;
- типів даних полів;
- описів полів (короткі коментарі);
- властивостей полів.

Розглянемо технологію створення цих елементів:

1) **Визначення імен полів.** Для визначення імені поля потрібно активізувати поле введення в стовпці “Ім'я поля” і ввести ім'я поля з клавіатури. Ім'я поля повинне бути унікальним (не повинне повторюватися в таблиці) і задовольняти угодам про імена об'єктів у *Access*: може складатися з букв, цифр, пробілів і спеціальних символів, за винятком символів: “.”(крапка), “!”, “‘”(апостроф), “[“, “]”. Використовувати пробіли не рекомендується.

2) **Визначення типів даних полів. Тип даних** визначає припустимі *значення*, які можна буде вводити в це поле, і *операції*, які можна виконувати з цими значеннями. У *Access* існують слідуючи 9 типів даних:

Короткій Текст – текст, довжиною не більш 255 символів;

Довгій Текст – – довгий текст, розміром не більш 64000 символів;

Числовий – числові значення, діапазон яких визначається властивістю “Розмір поля”;

Грошовий – числові значення, що містять 15 знаків у цілій частині і 4 – у дробовій (займає 8 байт);

Дата/Час – дата і час з роком від 100 до 9999 (8 байт);

Лічильник – у поле цього типу автоматично вводяться унікальні цілі числа, що збільшуються на 1. Значення поля не можна змінити чи видалити. Цей тип використовується в якості первинного ключа таблиці. Розмір автоматично встановлюється **Довге ціле;**

Логічний – може приймати одне з двох значень: “так”/”ні”, “Yes”/”No”, “істина”/ “неправда” і т.п. (займає 1 байт);

Об'єкт OLE – у полі міститься об'єкт (це може бути таблиця *Exel*, документ *Word*, малюнок, звуковий запис і т.д., яки *зв’язані* чи *впроваджені* у таблицю *Access*). Можлива довжина – до 1 Гбайта;

Входження – у цьому полі вказується шлях до файлу на жорсткому диску, шлях **UNC** чи адреса **URL**. Максимальна довжина – 64000 символів.

Розраховане - числові значення, визначається властивістю формулою.

Примітка: **UNC** – це формат мережного маршруту до файлу на сервері локальної мережі чи на жорсткому диску локального комп'ютера; **URL** – універсальний покажчик ресурсу (посилання) служби *www* у *Internet.*

Визначення типу даних для поточного поля виконується шляхом вибору потрібного типу зі списку, що розкривається. У розкритому стані список виглядає так, як це відображено на рис. 5.7.

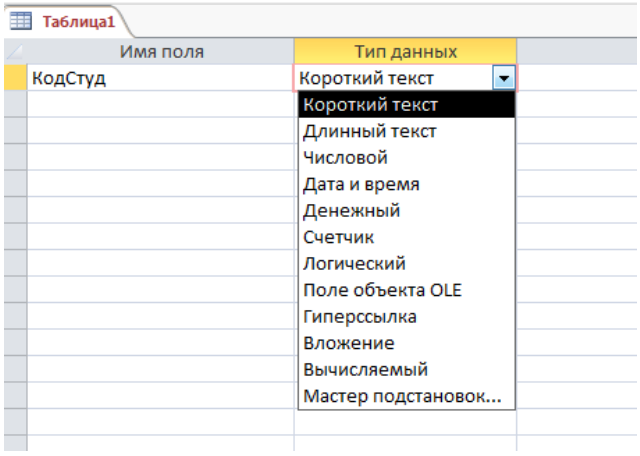


Рисунок 5.7. Список вибору типу даних поля у розкритому стані

3) **Визначення опису поля.** Опис – це короткий пояснювальний текст, що щораз буде з'являтися в рядку стану при виборі відповідного поля (у таблиці, запиті чи формі). Опис поля не є обов'язковим, однак при розробці БД рекомендується використовувати цю можливість, тому що опис є свого роду міні-довідкою, яка може виявитися дуже корисною для користувача. Уведення тексту опису виробляється в стовпці “Опис” у вікні Конструктора таблиць.

4) **Визначення властивостей полів.** Властивості полів залежать від типу даних. Список властивостей для обраного поля відображається в нижній частині вікна Конструктора таблиць на вкладці ”Загальні” (рис.5.8). Існують наступні властивості полів:

- розмір поля;
- формат поля;
- число десяткових знаків;
- маска введення;
- підпис;
- значення за замовчуванням;
- умова на значення;
- повідомлення про помилку;
- обов'язкове поле;
- порожні рядки;
- індексоване поле.

Розглянемо зміст і призначення властивостей полів.

Розмір поля визначає розмір даних, що зберігаються у цьому полі. Він залежить від типу даних наступним чином:

- для типу **Числовий** можна задати наступні розміри:
- **Байт** – цілі числа від 0 до 255, займає 1 байт;
- **Ціле** – цілі числа від –32.768 до +32.767, займає 2 байта;
- **Довге ціле** – цілі числа від –2.147.483.648 до +2.147.483.647, займає 4 байта;
- **Дрібне ціле з плаваючу крапкою 4 байта** – числа від $-3{,}4\times10^{38}$ до $+3{,}4\times10^{38}$ с точністю до 7 знаків;
- **Дрібне ціле з плаваючу крапкою 8 байт** – числа від $-1{,}797\times10^{308}$ до $+1{,}797\times10^{308}$ с точністю до 15 знаків;
- **Дійсне** – цілі числа від -10^{28}-1 до 10^{28}-1 с точністю до 28 знаків, займає 12 байт.

Для типу **Текстовий** розмір поля може бути заданий від 1 до 255 байтів (символів) (по замовченню 50 байт). Рекомендується задавати мінімальний розмір, який буде достатнім для збереження даних в цьому полі.

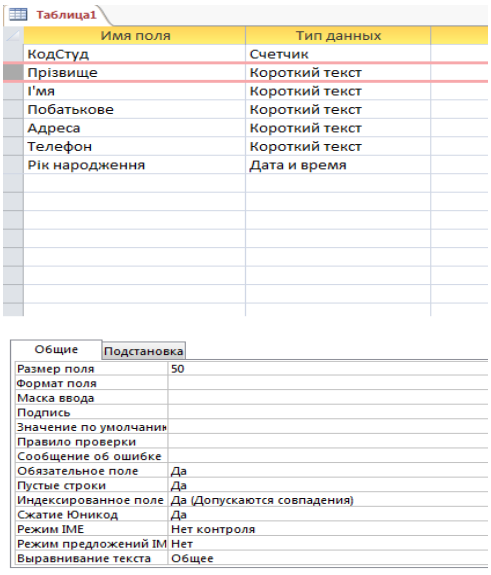


Рисунок 5.8. Список «**Общее**» для налаштування властивості поля з типом даних «Кротке текстове»

Формат поля – ця властивість встановлює правила, згідно з якими показуються дані при виведенні на екран чи до друку. Встановлені стандартні формати відображення для полів типів **Числовий**, **Дата/Час**, **Логічний**, **Грошовий**.

Кількість десяткових знаків задає для полів Числового і Грошового типів кількість знаків після коми. Можливо задавати значення від 0 до 15. Властивість впливає тільки на кількість десяткових знаків, яки відображуються на екрані і не впливають на кількість знаків, що зберігаються у пам'яті.

Маска введення – ця властивість дозволяє для типів даних **Текстовий**, **Грошовий**, **Дата/Час** відображати на екрані у ділянці уведення даних розподільні символи для полегшення (зменшення помилок) роботи користувача. Наприклад, для введення номера телефону можна створити маску (###) 000-00-00.

Підпис задає текст, який буде відображуватися у елементах керування (таблицях, формах, звітах).

Умова на значення задає обмеження на значення, яки можуть бути уведені у дане поле. При порушенні умови уведення даних забороняється і на екрані відображується текст, який задано у властивості **Повідомлення про помилку**.

Повідомлення про помилку задає текст повідомлення, який відображується на екрані при порушенні обмежень, заданих для цього поля.

Обов'язкове поле визначає, чи допустимо для даного поля не уводити ніякі дані.

Порожні рядки визначає, чи допустимо у дане поле уводити рядки розміром 0 символів. Ця властивість задається тільки для полів типа **Текстовий.**

Індексоване поле – ця властивість визначає, створювати чи ні для даного поля індекс. Можливо встановлювати такі варіанти цієї властивості:

Так (повторення не припускаються) – таке значення встановлюється для поля, яке повинно бути унікальним ключем. Для полів, які визначені як первинний ключ, таке значення встановлюється автоматично;

Так (повторення припускаються) – таке значення встановлюється для поля, яке повинно бути ключем зв'язку;

Ні – індекс для поля не створюється.

Для більшості з властивостей *Access* установлює значення за замовчуванням.

Описаним образом створюються всі поля таблиці. Після цього потрібно призначити первинний ключ.

5.3.2 Створення первинного ключа

Для створення первинного ключа необхідно виконати такі дії:

1) виділити поле, що повинне бути ключовим;
2) виконати команду меню **Ключевое поле** натиснув кнопку на панелі інструментів;

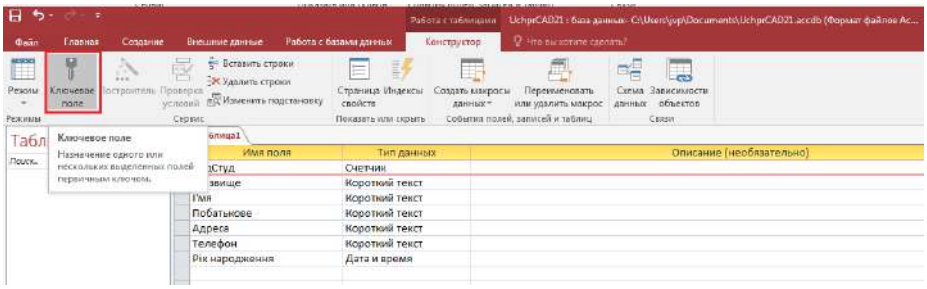


Рисунок 5.9 - Призначення ключового полю

Після цього ліворуч від імені виділеного поля (рис.5.9) з'явиться символ ключа в підтвердження того, що дане поле є ключовим (ключ створений) .

Якщо необхідно створити **складений первинний ключ** (що складається з декількох полів), то потрібно виконати всі ті ж дії для кожного з полів, утримуючи при цьому натиснутою клавішу ***Ctrl***.

5.3.3 Збереження структури таблиці

Збереження структури таблиці завжди відбувається після її створення чи після внесення змін у структурі таблиці при закритті вікна Конструктора таблиць. При цьому *Access* пропонує підтвердити необхідність зберегти таблицю чи зміни в її структурі. У діалоговому вікні, що при цьому відкривається, у відповідь треба увести “Так”. Якщо раніше таблиця не була збережена (тільки створена структура), то відкриється вікно, у якому потрібно ввести ім'я таблиці. Вікно має вигляд, показаний на рис. 5.10

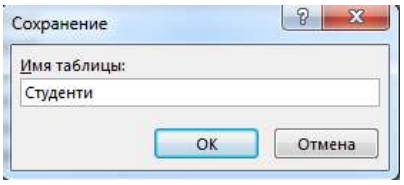


Рисунок 5.10 - Вікно для введення ім'я таблиці

Зберегти структуру створеної таблиці чи зміни в ній можна також, виконавши команду меню **Файл | Сохранить как...** .

Після створення таблиці у панелі **Таблиці** з'явиться відповідний їй ярлик (позначка) як це показано на рис. 5.11.

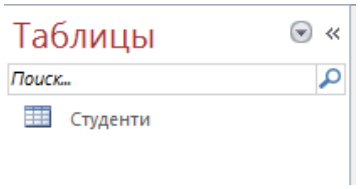


Рисунок 5.11 - Сторінка “Таблиці” вікна БД після створення таблиці СТУДЕНТИ

Контрольні запитання:

1. Призначення та можливості СУБД *Access*.
2. Основні етапи створення таблиць в *Access* за допомогою Конструктора.
3. Типи даних в *Access* та їх характеристики.
4. Властивості полів в Access та їх характеристики.
5. Як створити ключове поле?
6. Як створити складне ключове поле?
7. Правіла іменування полів таблиць.
8. Як створити зв’язок між таблицями один до одного?
9. Як створити маску поля?
10. Як створити поле що розраховують?

6. Поняття цілісності даних. Поля підстановки і фільтрація даних

6.1. Поняття цілісності даних. Зв'язування таблиць у Access

6.1.1. Поняття цілісності даних

Під цілісністю даних (посилальною цілісністю даних) у загальному значенні розуміють вимоги несуперечності представленої в БД інформації, щоб в БД не було посилань на неіснуючі дані, і не було зв'язаних даних, посилання на які відсутні (загублені дані).

Основним засобом забезпечення цілісності даних є створення зв'язків між таблицями БД. Зв'язки між таблицями відповідають зв'язкам між відповідними сутностями і відношеннями, дані про які представлені в цих таблицях. Так само, як і зв'язки між відношеннями, зв'язки між таблицями можуть бути зв'язками типу "1:1", "1:М" і "М:М". Зв'язок між таблицями встановлюється через зовнішні ключі (ключі зв'язку). У зв'язаних таблицях у зв'язаних записах значення полів, що є ключами зв'язку, повинні співпадати.

Розглянемо приклад зв'язаних таблиць. На рис. 6.1 показано дві таблиці: ТОВАРИ і ПОКУПКИ, між якими встановлений зв'язок типу "1:М".

ТОВАРИ

КоТов	*НазваТовару*	*ОдВиміру*	*ЦенаОд*
1	Цемент	кг	
2	Шифер	кв.м.	
3	Щебінка	т	
… … .	……	… …..	…… ..

ПОКУПКИ

КодПокуп	*КодТов*	*Дата*
1	1	2.10.02
2	3	2.10.02
3	3	3.11.02
4	1	3.11.02
……	…	…

Рисунок 6.1. Зв'язок типу "1:М" між таблицями ТОВАРИ і ПОКУПКИ

Зв'язок створено по полю (ключу зв'язку) **КодТов**. У таблиці ТОВАРИ поле **КодТов** є первинним ключем і одночасно ключем зв'язку. У таблиці ПОКУПКИ поле **КодТов** є ключем зв'язку, а первинним ключем є поле **КодПокуп**. У даному випадку прийнято таблицю ТОВАРИ називати *головною* таблицею, а таблицю ПОКУПКИ – *підлеглою*.

Цілісність даних забезпечується виконанням наступних вимог:

- У підлеглу таблицю не можна додати запис з неіснуючим у головній таблиці значенням ключа зв'язку.
- У головній таблиці не можна видалити запис, якщо в підлеглій таблиці є зв'язані з нею записи.
- Не можна змінити значення ключа зв'язку, якщо в зв'язаній таблиці є зв'язані записи.

Зв'язаними називають такі записи, у яких ключі зв'язку мають однакові значення.

У розглянутому прикладі цілісність даних буде порушена, якщо з таблиці ТОВАРИ видалити запис товару "Цемент", чи додати в таблицю ПОКУПКИ запис для якого-небудь товару, для якого ще не створений запис у таблиці ТОВАРИ. Цілісність даних також була б порушена, якщо в таблиці ПОКУПКИ змінити значення в поле **КодТов.**

Якщо користувач спробує виконати дії, що порушують цілісність даних, СУБД не дозволить це зробити і видасть повідомлення про неприпустимість даної дії.

В усіх СУБД (і в *Access*) маються спеціальні засоби для автоматизації деяких операцій по підтримці цілісності даних. Так, у СУБД *Access* передбачена можливість при створюванні зв'язків між таблицями задати вимогу автоматичного виконання таких операцій:

– При зміні значень у полях зв'язку головної таблиці автоматично змінювати значення зв'язаних полів у підлеглих таблицях (каскадне відновлення зв'язаних полів);

– При видаленні запису в головній таблиці автоматично видаляти зв'язані записи в підлеглих таблицях (каскадне видалення зв'язаних записів).

6.1.2. Створення зв'язків між таблицями в Access

Зв'язування таблиць у *Access* виконується шляхом побудови **Схеми даних** у такий спосіб. Відкрийте вікно “Робота з БД” та натиснуть кнопку  на панелі інструментів (рис.6.2).

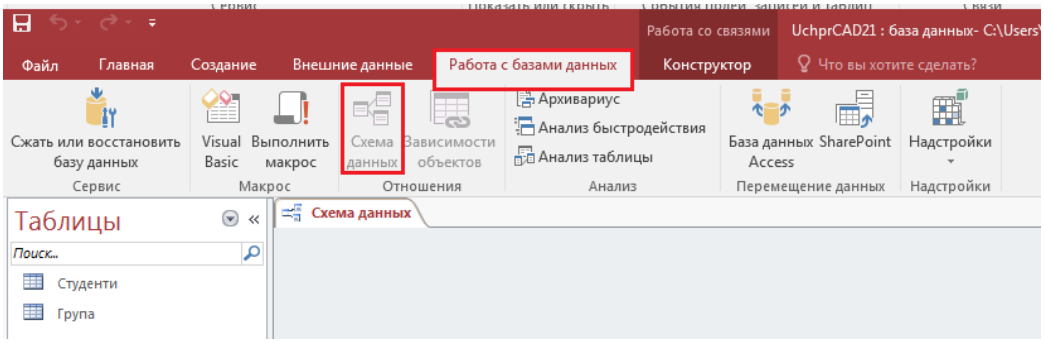


Рисунок 6.2 - Вікно схеми даних

У вікно “Схема даних” (на початку воно порожнє) треба помістити таблиці, між якими будуть установлюватися зв'язки. Для додавання таблиць потрібно натиснути праву кнопку миші та визвати контекстне меню у якому вибрати строку Додати таблицю (рис.6.3).

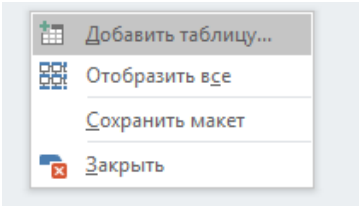


Рисунок 6.3 - Вікно додавання таблиць у схему даних

Відкриється вікно "Додавання таблиці" (при створенні Схеми даних це вікно відкривається відразу), вид якого показаний на рис. 6.4.

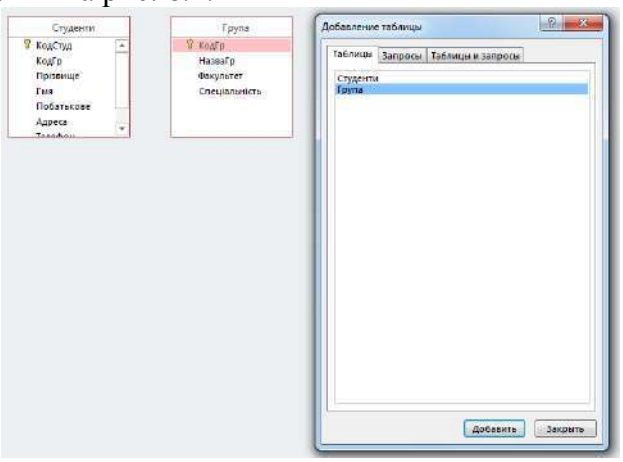


Рисунок 6.4- Вікно “Додавання таблиць”

У цьому вікні потрібно послідовно вибрати потрібні таблиці і додати їх у Схему даних натисканням кнопки **Додати.** Після додавання таблиць натисніть **Зачинити**.

Для встановлення зв'язку необхідно виділити курсором миші поле первинного ключа в списку полів однієї таблиці (наприклад, ГРУПИ) і утримуючи ліву кнопку натиснутою перемістити курсор миші на поле, з яким установлюється зв'язок, у списку полів другої таблиці (наприклад, СТУДЕНТИ).

Після відпускання лівої кнопки миші на екрані з'явиться вікно “Зміна зв'язку” (рис. 6.5), у якому потрібно задати параметри властивостей зв’язку.

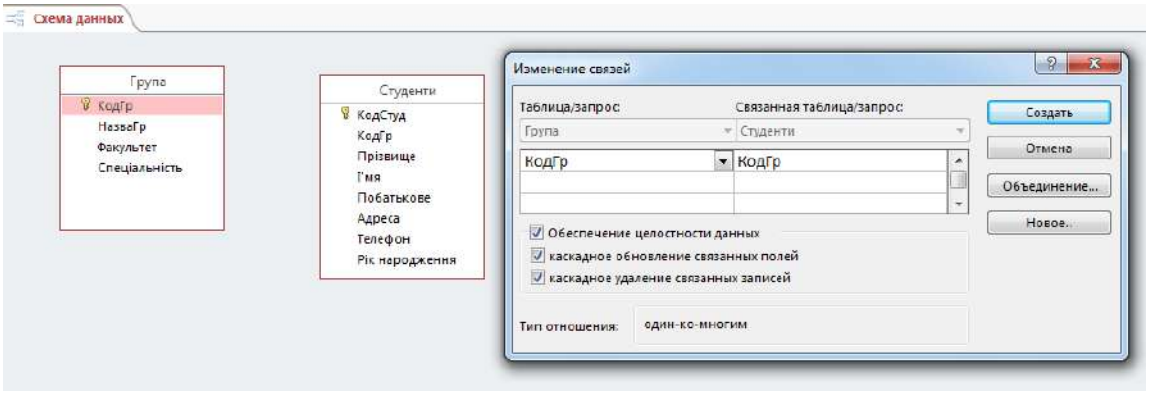


Рисунок 6.5 - Вікно “Зміна зв’язку”

У верхній частині вікна відображуються поля, які будуть полями зв’язку. Нижче розташовано перемикач “Забезпечення цілісності даних”. Завжди рекомендується включати цей перемикач. Для цього достатньо створити на ньому щиглик мишею. Після цього стануть активними ще два перемикачі: "каскадне відновлення зв'язаних полів", і "каскадне видалення зв'язаних записів". За допомогою перемикачів задайте потрібні параметри забезпечення цілісності даних.

У нижній частині вікна відображується тип зв’язку, який встановлюється. Тип зв’язку *Access* визначає автоматично по типу даних полів, які вибрані як ключі зв’язку.

Після встановлення потрібних параметрів натисніть кнопку **Створити**. У вікні “Схема даних” з'являться стрілки, що з'єднують таблиці і, що вказують на тип зв'язку. Схема даних отримає приблизно такий, як показано на рис. 6.6.

Для зручного розташування списків полів таблиць на Схемі даних ви можете переміщувати таблиці, утримуючи натиснутою ліву кнопку миші в ділянці заголовка таблиці. Зв'язки між таблицями створені. Закрийте вікно “Схема даних”.

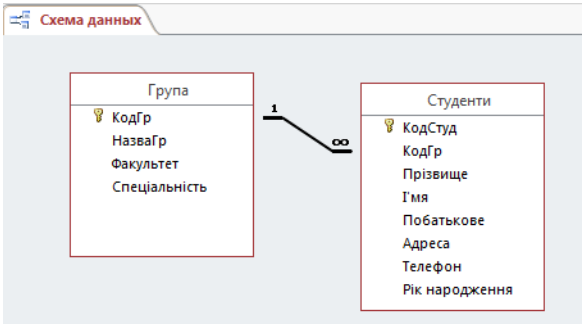


Рисунок 6.6 - Вид Схеми даних після створення зв’язків між таблицями

6.2. Створення полів підстановок у Access

6.2.1. Ідея створення полів підстановки

При роботі з даними (при перегляді, введенні чи редагуванні даних) дуже незручно користатися кодами, що не мають якого-небудь змістовного значення. Наприклад, у таблиці УСПІШНІСТЬ у поле **КодСтуд** потрібно вводити коди студентів, яки для користувача не мають розуміння, і знати їх йому зовсім не обов'язково (і не потрібно). Користувачу замість коду необхідно відображати прізвище студента.

Для рішення цієї проблеми використовуються так називані поля підстановки (*Lookup*-поля). Ідея полягає в тому, що в таблиці створюється додаткове поле – поле підстановки, у яке містяться значення, узяті зі зв'язаної таблиці. На рис. 6.7 у таблиці УСПІШНІСТЬ створюється поле підстановки **ПІБ** (прізвище, ім'я, по батькові) , у яке міститься значення з поля **ПІБ** зв'язаного запису таблиці СТУДЕНТИ.

Зв'язаний запис, з якого виконується підстановка, визначається за значенням ключа зв'язку **КодСтуд.**

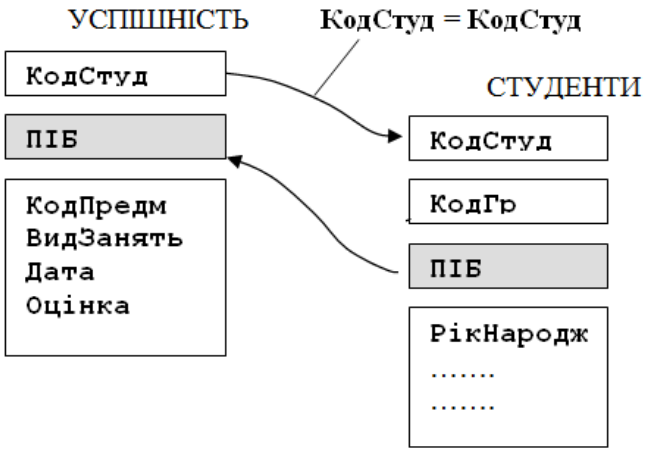


Рисунок. 6.7 - Схема створення поля підстановки **ПІБ**

Існує й інший тип поля підстановки – підстановка зі списку фіксованих значень. Необхідність у такої підстановки виникає в тих випадках, коли в якесь поле потрібно вводити повторювані значення.

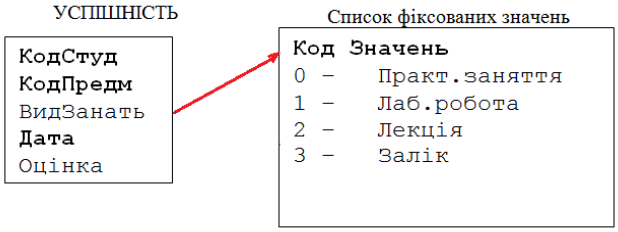


Рис. 6.8 - Створення поля підстановки з фіксованого списку

Наприклад, у таблиці УСПІШНІСТЬ у полі **ВидЗанят** приходиться вводити повторювані значення з клавіатури. Набагато зручніше в цьому випадку створити список фіксованих значень, наприклад: “Практ. заняття”, “Лаб. робота”, “Лекція” і ін., і потім уводити потрібне значення шляхом вибору з цього списку. Завдяки цьому значно прискорюється процес введення і виключаються помилки, що були б неминучі при введенні даних з клавіатури.

При створенні поля підстановки з фіксованого списку автоматично формуються внутрішні коди для елементів списку, що зберігаються в цьому полі і використовуються при введенні і відображенні даних (рис. 6.8).

Далі розглянемо технологію створення полів підстановки в СУБД *Access*.

6.2.2. Створення полів підстановки зі зв'язаної таблиці (чи запиту)

Технологію створення полів підстановки зі зв'язаної таблиці (чи запиту) будемо розглядати на конкретному прикладі. Нехай потрібно створити в таблиці УСПІШНІСТЬ поле підстановки **Прізвище,** дані для якого отримуються зі зв’язаної таблиці СТУДЕНТИ.

Для створення поля підстановки потрібно виконати наступне:

Відкрити в режимі Конструктора таблицю (наприклад, це УСПІШНІСТЬ) і виділити в поле, для якого буде створюватися властивість підстановки (це поле **КодСтуд),** стовпчик "Тип даних".

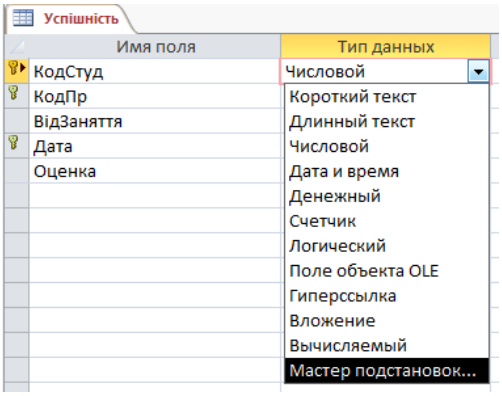


Рисунок 6.9 - Вибір майстру підстановки

За допомогою кнопки розкрити список типів даних і вибрати в ньому рядок (рис.6.9) "Майстер підстановок…". Відкриється перше вікно Майстра підстановок, вид якого показаний на рис. 6.10.

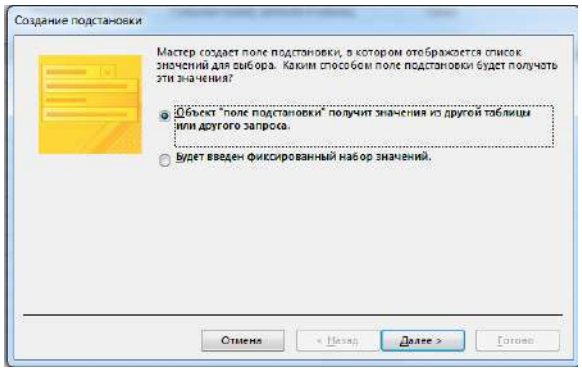


Рисунок 6.10 - Вибір способу формування поля підстановки

У цьому вікні виберіть перемикач "стовпець підстановки буде використовувати значення з таблиці чи запиту" (він пропонується за замовчуванням) і натисніть кнопку **Далі.** Відкриється вікно, показане на рис. 6.11. У цьому вікні відобразиться список таблиць і запитів. Виберіть у списку таблицю, з якої будуть братися дані для поля підстановки (виберіть таблицю **СТУДЕНТИ**) і натисніть кнопку **Далі.**

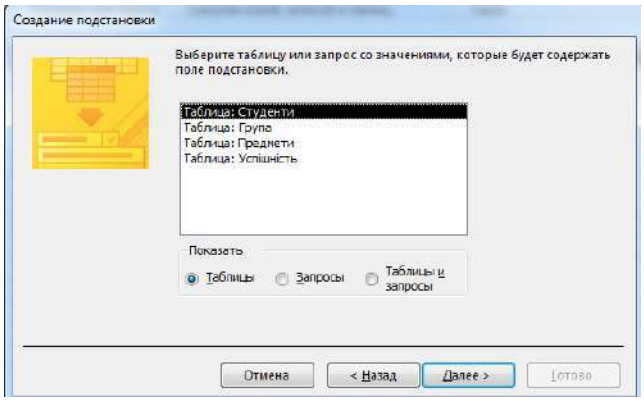


Рисунок 6.11- Вибір таблиці, з якої будуть братися дані для поля підстановки

У наступному вікні треба вибрати поля таблиці які будуть відображатися у списку підста-

новки (рис.6.12).

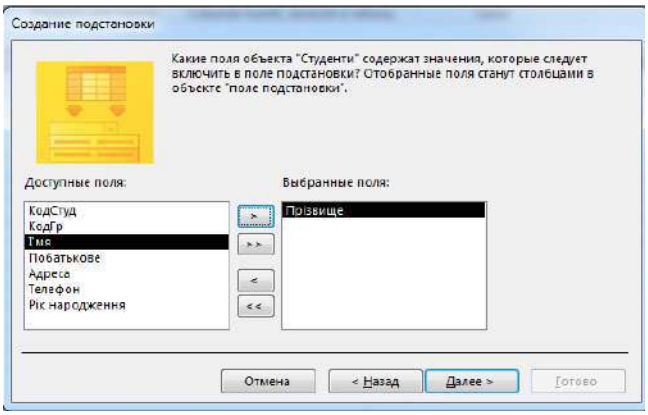


Рисунок 6.12 - Вибір полі, значення котрого буде міститися в поле підстановки

У наступному вікні можна відредагувати ширину стовпця. За допомогою перемикача “Скрити ключовий стовпчик” можна включити чи виключити відображення ключового стовпця. По замовченню рекомендується оставити його у включеному стані. Потім знову натисніть кнопку **Далі,** після чого відкриється завершальне вікно (рис. 6.13).

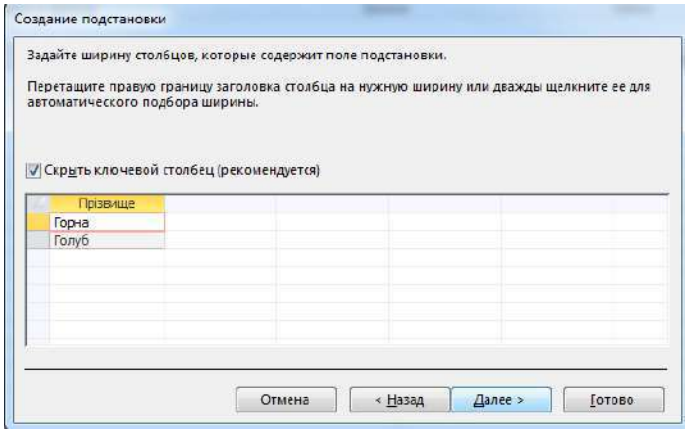


Рисунок 6.13 - Відображення даних, що будуть показуватися у полі підстановки

У завершальному вікні можна задати підпис (заголовок) для стовпця підстановки. Натисніть кнопку **Готове**. Вікно Майстра підстановок закриється.

Закрийте вікно Конструктора таблиць зі збереженням зроблених змін у структурі таблиці. Поле підстановки створене.

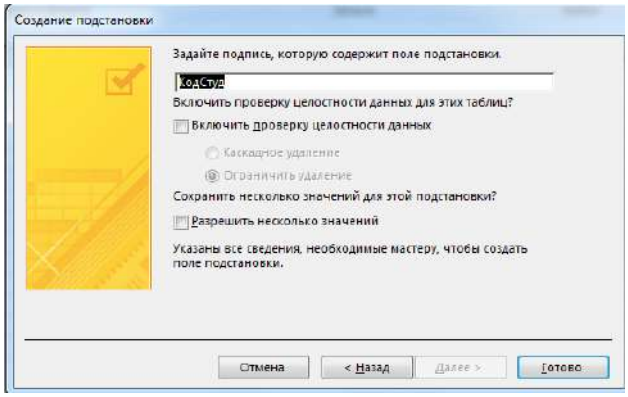


Рисунок 6.14 - Завершальне вікно Майстра підстановки

Тепер можна відкрити таблицю УСПІШНІСТЬ у режимі Таблиці і переконатися, що в по-

лі **КодСтуд** замість числового коду показуються прізвища студентів. При виборі якого-небудь запису в цьому полі праворуч з'являється кнопка , при натисканні якої відкривається список прізвищ усіх студентів, дані про які уже введені в таблиці СТУДЕНТИ (рис. 6.15).

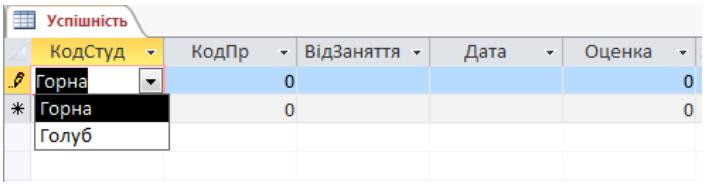


Рисунок 6.15 - Уведення даних у поле підстановки шляхом вибору зі списку значень, узятих зі зв'язаної таблиці

6.2.3. Створення поля підстановки з фіксованого списку значень

Тепер розглянемо технологію створення поля підстановки з фіксованого списку. Наприклад, для того, щоб у поле **ВидЗанят** таблиці УСПІШНІСТЬ щораз не вводити з клавіатури одні і ті ж самі слова: “Лекція”, “Лаб_робота”, “Залік” і т.д., створимо для цього поля підстановку з фіксованого списку. Для цього потрібно виконати наступне:

1) точно так само відкрийте таблицю УСПІШНІСТЬ у режимі Конструктора і для поля **ВидЗанят** викличте "Майстер підстановок…".

2) у першому вікні Майстра підстановок тепер виберіть пункт "буде уведений фіксований набір значень" (рис. 6.16). Після цього натисніть кнопку **Далі.**

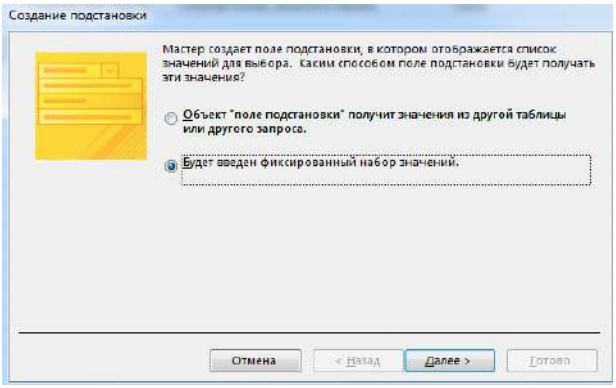


Рисунок 6.16 - Вибір способу створення підстановки з фіксованого списку

Відкриється вікно з таблицею (спочатку порожній), у яку потрібно увести потрібні фіксовані значення. Тут же можна задати число стовпців списку (за замовчуванням пропонується 1). Наприклад, можна ввести значення так, як це показано на рис. 6.17. Потім знов натисніть кнопку **Далі.**

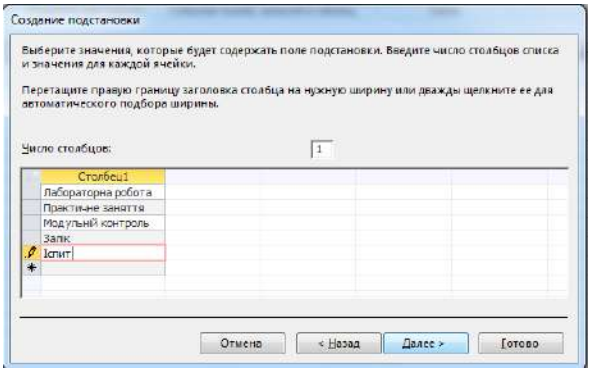


Рисунок. 6.17 - Створення списку фіксованих значень

Відкриється завершальне вікно Майстра, таке ж, яке було показано раніше на Рис. 6.18. Натисніть кнопку **Готове.**

Закрийте вікно Конструктора таблиць зі збереженням змін, які були виконані. Поле підстановки з фіксованого списку створено.

Тепер при виділенні поля **ВидЗанят** з'являється кнопка відкриття списку, що випадає. Якщо клацнути мишею на цій кнопці, то відкриється список, у якому будуть міститися введені раніше фіксовані значення (див. рис. 6.18).

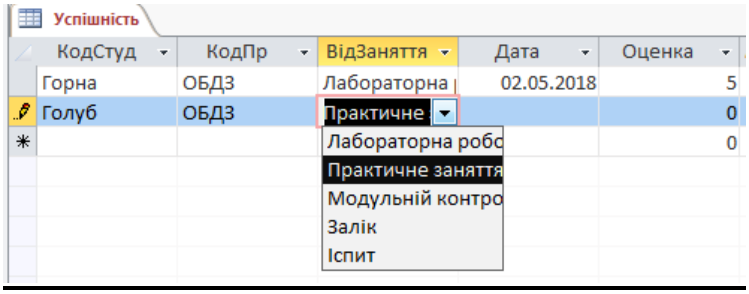


Рисунок 6.18 - Уведення даних у поле підстановки зі списку фіксованих значень

6.3. Фільтрація даних у Access

Фільтри - це зручний засіб відображення тільки тих даних, що у даний момент потрібні користувачу. У *Access* є три різновиди фільтрів: фільтр по виділеному, простий і розширений фільтри. Розглянемо коротко кожну з цих різновидів фільтрів.

6.3.1. Фільтр по виділеному

Фільтр по виділеному - це найбільш простіший спосіб добору даних. Для створення фільтра по виділеному необхідно:

- відкрити таблицю в режимі Таблиці;
- виділити курсором миші потрібний елемент даних;

1) натиснути кнопку  (“Виділення”) на панелі інструментів.

У результаті фільтрації в таблиці будуть відображатися тільки ті записи, які містять дані, що збігаються з виділеними (рис.6.19).

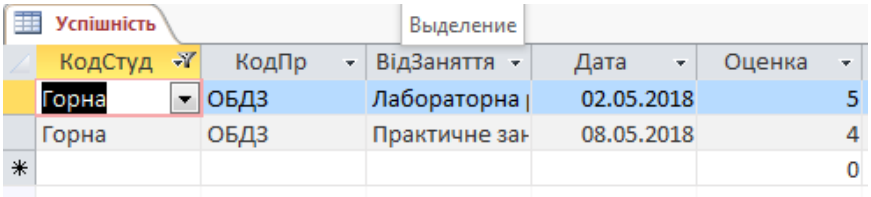
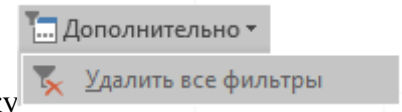


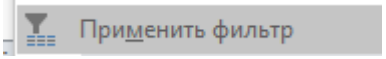
Рисунок 6.19 - Фільтр по виділеному

Кожне нове виділення накладає додаткові умови на фільтрацію даних.

Можна здійснювати фільтрацію за умовою “не міститься виділене значення”. Для цього потрібно виконати команду **Сортування і фільтр |Фільтр | Не містить виділене.**

Скасувати дію фільтра (і знов побачити всі записи таблиці) можна за допомогою команди **Сортування і фільтр | Додатково | Видалити фільтри** чи натиснувши кнопку 

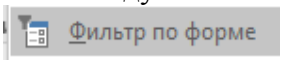
(“Видалити фільтр”).

Фільтр можна в будь-який момент застосувати повторно, виконавши команду меню **Сортування і фільтр | Застосувати фільтр** чи натиснувши кнопку 

("Застосувати фільтр").

Зверніть увагу, що ця кнопка фіксується в двох положеннях: у натиснутому стані - це кнопка "Видалити фільтр", у віджатому – "Застосувати фільтр".

6.3.2. Звичайний фільтр

Для створення звичайного фільтра необхідно виконати команду меню **Сортування і фільтр | Додатково | Фільтр по формі** чи натиснути кнопку 

.

На екрані відкриється вікно для введення і редагування умов фільтрації (рис. 6.20). У верхній частині вікна розташований рядок з іменами всіх полів обраної таблиці і під нею рядок, що містить задані умови фільтрації. Цей рядок буде містити умови, що відповідають поточному збереженому фільтру. У приведеному прикладі задана умова відбору студентів із прізвищем на "Голуб".

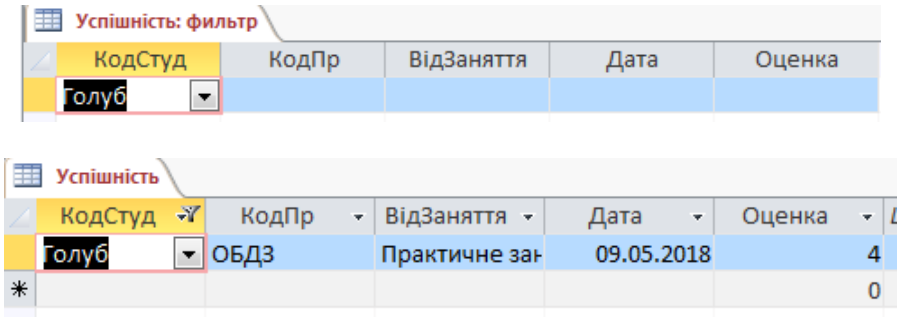
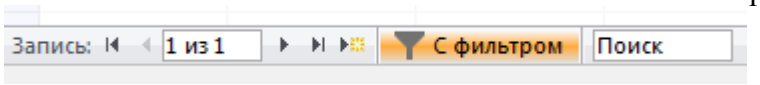


Рисунок 6.20. Вікно для введення умов фільтрації

У нижній частині вікна розташовані ярлички вкладок 

"З фільтром" і " Пошук ".

Вкладка Пошук" призначена для введення додаткових умов фільтрації даних.

Умови відбору можуть уводитися з клавіатури чи зі списку вибору, що відкривається при натисканні кнопки у відповідному полі.

Умови, введені в одному рядку для декількох полів розглядаються як об'єднані операцією "і" ("and"). Наприклад, для добору всіх студентів із прізвищами на "А" 1997 року народження потрібно ввести для поля **Прізвище** умову "А*" і для поля **Рік_Народж** - 1997.

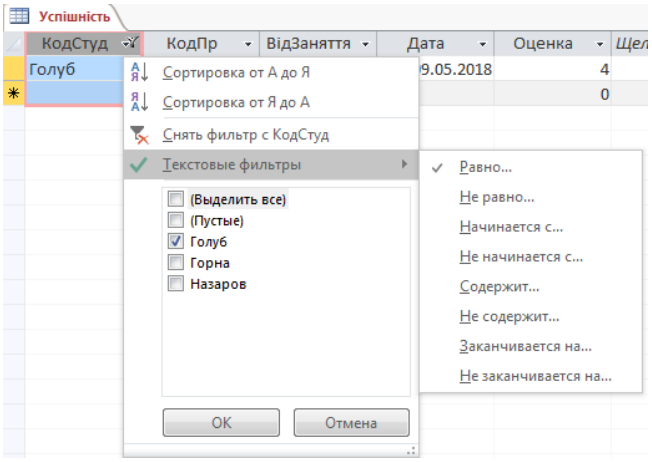
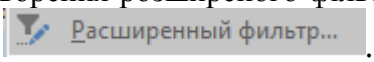


Рисунок 6.21 - Вибір текстового фільтру

Для введення додаткових умов добору необхідно вибрати вкладку "Чи" і в рядку на цій вкладці ввести додаткові умови. Ці умови будуть зв'язуватися з умовами попередньої сторінки (вкладки) операцією "Чи" ("or").

Для застосування фільтра, як і раніше, необхідно виконати команду **Застосувати фільтр** чи натиснути кнопку  .

6.3.3. Розширений фільтр

Для створення розширеного фільтра необхідно виконати команду **Додатково | Розширений фільтр** 

.

У результаті виконання команди відкриється вікно розширеного фільтра, можливий вид якого зображено на рис. 6.22. У верхній частині вікна відображається список полів поточної активної таблиці.

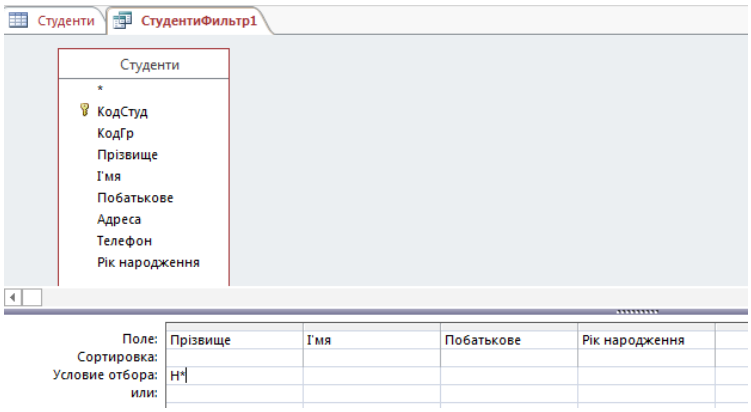


Рисунок 6.22 - Вікно розширеного фільтру

Крім того, якщо для якихось полів створені підстановки і зв'язані поля беруть участь в умовах добору, то відображаються також списки полів зі зв'язаних таблиць. У нижній частині вікна відображається бланк запиту, у якому задаються умови відбору записів.

У рядку "Поле" уводяться найменування полів, що будуть відображатися в результуючому (відфільтрованому) наборі даних. Уводити поля можна способом перетаскування зі списку у верхній частині вікна, чи способом вибору зі списку, що відкривається, наприклад, так, як це показано рис. 6.23.

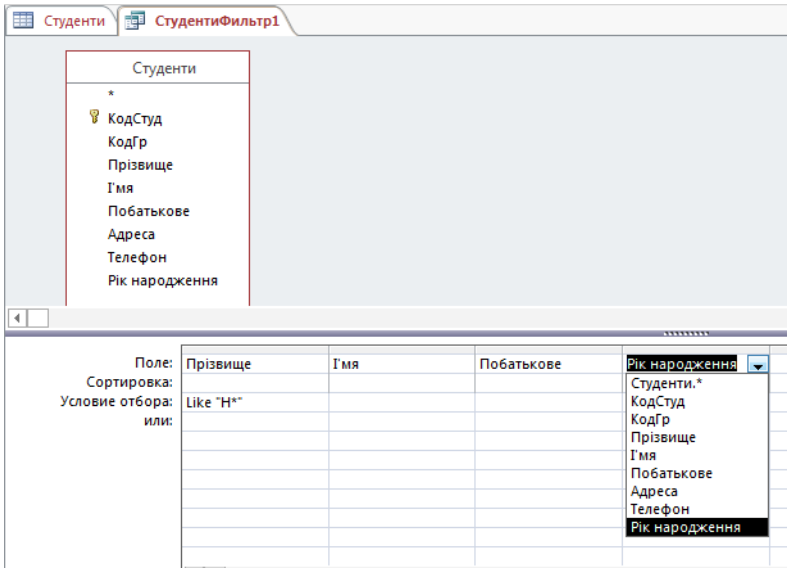


Рисунок 6.23 - Введення поля для розширеного фільтру шляхом вибору зі списку

У рядку "Сортування" можна вибрати тип сортування для одного чи декількох полів.

У рядку “Умова відбору” задається умова фільтрації так саме, як це було розглянуто для звичайного фільтра.

Наприклад, якщо потрібно відібрати всіх студентів із прізвищем на "А" 1997 і 1998 року народження, то на додаток до введених раніше умов необхідно ввести на вкладці "Чи" умови: для поля **Прізвище** - "Н*", для поля **Рік_Народж** - 1997.

При завданні умов фільтрації можна уводити вираження з використанням операцій: >, <, <>, and, or, not.

Наприклад, для одержання того ж результату можна на сторінці "Знайти" для поля **Рік_Народж** увести вираження: 1999 or 1998.

Для застосування фільтра, як і раніше, необхідно натиснути кнопку  (“Застосувати фільтр”).

6.3.4. Збереження і знищення фільтра

Створені фільтри (фільтр по виділеному, звичайний і розширений фільтри) завжди зберігаються автоматично при збереженні таблиці. При повторному відкритті таблиці збережений фільтр є поточним і може бути виконаний по команді Додатково | **Застосувати фільтр**.

Якщо створюється новий фільтр, він заміняє фільтр, раніше збережений з даною таблицею.

Для знищення фільтра необхідно на панелі інструментів натиснути кнопку  (“Змінити фільтр”) і потім - кнопку  (“Очистити бланк”). Після цього натиснути кнопку  (“Застосувати фільтр”). Фільтр, збережений з таблицею, буде знищений.

Контрольні запитання:

1. Поняття цілісності даних.
2. У яких випадках цілісність даних буде порушуватися?
3. Створення зв’язку між таблицями в *Access*.
4. Ідея полів підстановки.
5. Створення полів підстановки в *Access*.
6. Поняття фільтра.
7. Види фільтрів в *Access* і їх застосування.
8. Збереження та знищення фільтрів в *Access*.

7. Мова SQL

7.1. Загальні відомості

Запити – це основний інструмент вибірки й обробки даних у СУБД. Для створення і реалізації запитів розроблена спеціальна мова *SQL* (*Structured Query Language* – мова структурованих запитів).

Перший прототип мови *SQL* з'явився наприкінці 70-х років і одержав через якийсь час широке поширення. Він став застосовуватися у всіх комерційних СУБД і поступово став стандартом де-факто для мов маніпулювання даними в реляційних БД.

Перша версія стандарту називалася *SQL-86* і була прийнята *ANSI* (Американським національним інститутом стандартів) і *ISO* (Міжнародним інститутом стандартів). Наприкінці 2011 р. був прийнятий новий міжнародний стандарт мови – це *SQL-2011*. Цей стандарт підтримується всіма сучасними СУБД (табл.7.1).

Таблиця 7.1 Міжнародні стандарти мови SQL

Рік	Назва	Інша назва	Коментар
1986	SQL-86	SQL-87	Вперше оприлюднено ANSI. Ратифіковано ISO в 1987.
1989	SQL-89	FIPS 127-1	Незначні зміни.
1992	SQL-92	SQL2	Вагомі зміни.
1999	SQL:1999	SQL3	Додано регулярні вирази, рекурсивні запити, тригери та деякі об'єктно-орієнтовані нововведення.
2003	SQL:2003	SQL 2003	Впроваджені розширення для роботи з XML-даними.
2006	SQL:2006	SQL 2006	ISO/IEC 9075-14:2006. Функціональність роботи з XML-даними значно розширено. З'явилась можливісь сумісного використання в SQL та XQuery.
2008	SQL:2008	SQL 2008	Вдосконалені можливості віконних функцій, усунуто деякі неоднозначності стандарту SQL:2003. Легалізовано ORDER BY поза визначенням курсору. Додано тригери INSTEAD OF. Додано заяви TRUNCATE.
2011	SQL:2011	SQL 2011	ISO/IEC 9075:2011.

Ми коротко ознайомимося з деякими найбільш важливими елементами мови.

Всі оператори мови можна розділити на такі три категорії:

Оператори контролю даних – використовуються для перевірки повноважень користувача при звертанні до БД. Наприклад, це такі оператори:

GRANT – надати права доступу до тих чи інших об’єктів БД чи функцій;

REVOKE – позбавити тих чи інших прав.

Оператори визначення даних – використовуються для створення об'єктів БД і визначення їхньої структури. До них відносяться, наприклад, такі оператори:

CREATE TABLE – створити таблицю;

DROP TABLE – видалити таблицю;

ALTER TABLE – змінити структуру існуючій таблиці;

CREATE INDEX – створити інде, і інші оператори;

Оператори керування даними – використовуються для пошуку, видалення, зміни і збереження даних. Це такі оператори:

SELECT – оператор, який реалізує усі операції реляційної алгебри і дозволяє створювати нові відношення згідно запиту, що був заданий;

UPDATE – відновлює значення одного чи декількох полів в однієї чи декількох рядках таблиці;

INSERT – утворює новий запис і додає його в таблицю;

DELETE – видаляє з таблиці один чи декілька записів, які відповідають умовам, що задані.

На початку вивчення мови *SQL* доцільно розглянути оператор SELECT, який найчастіше використовується на практиці. Його вивчення дозволить глибше зрозуміти внутрішні механізми функціювання БД.

7.2. Використання операторів DDL

Оператори DDL (Data definition language - мова опису даних) дозволяють створювати, модифікувати і видаляти в базі даних таблиці і індекси. За допомогою цих операторів багато які досить складні операції можна записати в одному рядку. Проте, є і деякі обмеження. Головне з них полягає в тому, що оператори DDL підтримуються тільки базами даних Jet. Нагадаємо, що об'єкти доступу до даних можуть використовуватися з будь-якими базами даних, що підтримуються ядром Jet. Інше обмеження підтримка тільки невеликої підмножини властивостей об'єктів таблиць, полів і індексів. Якщо необхідно працювати з властивостями поза цією підмножиною, доведеться скористатися методами, описаними раніше.

Для опису таблиць в базах даних використовуються три оператори DDL:

- CREATE TABLE. Визначає в базі даних нову таблицю.
- ALTER TABLE. Змінює структуру таблиці.
- DROP TABLE. Видаляє таблицю з бази даних.

Для створення таблиці необхідно побудувати SQL-оператор, що містить її ім'я і імена, типи і розміри всіх полів.

У наступному прикладі створюється таблиця Orders:

```
CREATE TABLE Orders (
Orderno LONG,
Custno LONG,
SalesID TEXT(6),
OrderDate DATE,
Totcost SINGLE);
```

Зверніть увагу, що при вказівці імен таблиці і полів, їх не треба брати в лапки. Імена, що містять пропуски, треба брати в квадратні дужки (наприклад, [Last name]).

При створенні таблиці вказуються тільки імена, типи і розміри полів. Не можна указати додаткові параметри, такі як значення, задані за умовчанням, перевірочні правила або перевірочні повідомлення про помилки. Але навіть з цими обмеженнями оператор DDL CREATE TABLE виявляється засобом створення таблиць.

Скориставшись оператором ALTER TABLE, можна додати в існуючу таблицю нове поле або видалити з неї старе. При доданні поля необхідно указати його ім'я, а також його тип і розмір (при необхідності). Для додання полів застосовується директива ADD COLUMN оператора ALTER TABLE. Для видалення поля досить указати його ім'я в директиві DROP COLUMN оператора ALTER TABLE. Як і при використанні інших методів зміни структури таблиць, в цьому випадку не можна видалити поле, що застосовується в індексі або відношенні.

У лістинг демонструється додання і видалення полів в таблиці Orders, яку ми створили в попередньому розділі.

Додаємо в таблицю "Orders" поле для вартості доставки :

```
ALTER TABLE Orders ADD COLUMN Shipping SINGLE
```

Видаляємо поле для вартості доставки :

```
ALTER TABLE Orders DROP COLUMN Shipping
```

За допомогою оператора DROP TABLE з бази даних можна видаляти цілі таблиці. У наступному фрагменті коду віддаляється таблиця Orders. При видаленні цілих таблиць вимагайте підтвердження операції від користувача; відразу після виконання цього оператора таблиця і дані, що все містяться в ній будуть безповоротно видалені:

```
DROP TABLE Orders
```

Роботу з індексами забезпечують два оператори DDL:

- CREATE INDEX. Створює новий індекс для таблиці.
- DROP INDEX. Видаляє існуючий індекс.

За допомогою оператора CREATE INDEX можна створювати індекси на основі одного або декількох полів. При цьому необхідно указати ім'я індексу, ім'я таблиці, що індексується і, принаймні, одне поле, на основі якого будується індекс. Можна також указати порядок індексації (висхідний або низхідний) для кожного поля. Крім того, індекс можна зробити первинним. У лістинг створюється первинний індекс по номеру замовника і ще один індекс по двох полях із заданим порядком сортування. Індекси створюються для таблиці Customers.

Створення первинного індексу по номеру замовника:

CREATE INDEX Custno ON Customers (Custno) WITH PRIMARY

Створення індексу по двох полях: по LastName у висхідному і по FirstName в низхідному порядку:

CREATE INDEX Name2 ON Customers (Lastname ASC, FirstName DESC)

Позбутися індексу також просто, як і створити його. Досить скористатися оператором DROP INDEX, як показано в наступному прикладі. Приведені оператори видаляють обидва індекси, створені в попередньому лістинг. Зверніть увагу, що при видаленні індексів необхідно вказувати ім'я відповідної таблиці:

DROP INDEX Custno ON Customers DROP INDEX Name2 ON Customers

7.3. Оператор SELECT. Відбір записів з однієї таблиці

Загальний формат оператора SELECT наступний:

SELECT [DISTINCT] {* | ⟨Список полів⟩}
FROM ⟨Список таблиць⟩
[WHERE ⟨Умова добору записів⟩]
[GROUP BY ⟨Список полів для групування⟩]
[HAVING ⟨Умови добору для груп⟩]
[ORDER BY ⟨Список полів для сортування⟩]

У наведеному форматі дужки [] позначають, що даний елемент є необов'язковим. Дужки {} позначають множину елементів. Дужки ⟨⟩ позначають елемент, зміст якого пояснюється текстом в дужках.

Результатом виконання оператора SELECT є набір даних, який складається з записів, що відповідають заданим умовам відбору. В операторі обов'язково повинні бути присутніми інструкції SELECT і FROM. Інші інструкції (вказані у квадратних дужках) можуть бути відсутніми. Розглянемо більш докладно кожну з інструкцій, з яких складається оператор SELECT.

Інструкція SELECT повідомляє СУБД, що це початок оператору. Одночасно в інструкції SELECT указується список полів, які будуть включатися в записи, що відбираються. У списку полів повинне бути задане хоча б одне поле. Якщо в список полів потрібно включити всі поля з таблиці (таблиць), то замість перерахування полів можна вказати символ *. Якщо в список полів включаються поля з різних таблиць, то для вказівки приналежності поля до тієї чи іншої таблиці використовують складене ім'я поля, що складається з імені таблиці й імені поля, розділених крапкою. Необов'язкова інструкція DISTINCT забороняє включення в результуючий набір даних повторюваних записів.

В інструкції FROM вказуються імена таблиць, з яких відбираються записи. Список повинний містити хоча б одну таблицю. Якщо кількість таблиць дві і більше, між таблицями повинне встановлюватися з'єднання. Синтаксис інструкції у випадку з'єднання двох чи декількох таблиць буде розглянуто пізніше.

В інструкції WHERE задається умова (критерій) відбору записів, яка представляється логічним виразом. Логічний вираз складається з операндів, операцій порівняння і логічних операцій. У якості операндів можуть використовуватися імена полів і константи. У логічних виразах

для завдання умов відбору можуть використовуватися такі операції порівняння і логічні оператори і операції: = , < , > , <> , <= , >= – операції порівняння;

Between – предикат, що перевіряє приналежність значення поля заданому діапазону значень. Предикат – це логічна функція, яка приймає значення *True* чи *False* в залежності від значень параметрів функції-предиката;

In – предикат, що перевіряє приналежність значення поля заданій множині;

Like – предикат, що перевіряє відповідність значення поля заданому шаблону;

And, Or, Not – логічні операції.

Інструкція GROUP BY призначена для вказівки полів, по яких визначаються групи записів. В одну групу включаються записи з однаковими значеннями в полях, перерахованих в інструкції GROUP BY. Для груп записів можна застосовувати групові операції (їх ще називають агрегатними функціями). У мові *SQL* визначені такі групові операції:

Max( ) – вибирає максимальне значення поля;

Min( ) – вибирає мінімальне значення поля;

Count( ) – визначає число значень у групі;

Avg( ) – обчислює середнє значення;

Sum( ) – обчислює суму значень полів у групі.

В дужках вказується ім'я поля, к даним якого застосовується дана функція.

Якщо застосовується інструкція GROUP BY, то у результуючому наборі даних кожний запис представляє одну з груп, що визначені в інструкції.

Інструкція HAVING застосовується разом з інструкцією GROUP BY і використовується для завдання умов добору для згрупованих даних. Правила запису умов добору аналогічні правилам завдання умов в інструкції WHERE.

В інструкції ORDER BY указується список полів, по яких потрібно сортування записів у результуючому наборі даних. За замовчуванням сортування по кожному полю виконується в порядку зростання значень. Якщо необхідно зробити сортування по спаданню, то після імені відповідного поля потрібно записати покажчик DESC.

Розглянемо приклади оператору SELECT для відбору записів з однієї таблиці:

```
SELECT  ПІБ, РікНародж, Адреса
FROM  СТУДЕНТИ
```

Цей оператор у результуючу таблицю відбирає поля **ПІБ** (прізвище, і'мя, по батькові), **РікНародж** і **Адреса** з таблиці СТУДЕНТИ.

```
SELECT  *
FROM  СТУДЕНТИ
```

Цим оператором відбираються всі поля з таблиці СТУДЕНТИ.

```
SELECT  DISTINCT ПІБ
FROM  СТУДЕНТИ
```

Необов'язкове ключове слово DISTINCT виключає відбір однакових записів.

```
SELECT  DISTINCT *
FROM  СТУДЕНТИ
```

Тут слово DISTINCT не має змісту, тому що виводиться ключове поле, значення в котрому не можуть повторюватися.

```
SELECT  ПІБ
FROM  СТУДЕНТИ
WHERE РікНародж >= 1999
```

Цим оператором відбираються записи з прізвищами усіх студентів тільки 1980 року народження і молодше.

```
SELECT  ПІБ
FROM  СТУДЕНТИ
WHERE (РікНародж = Between 1980 AND 1984) AND
        (Місто = “Київ”)
```

Відбираються записи, що містять поле **ПІБ**, для всіх студентів, рік народження яких належить діапазону від 1980 до 1984 років, і проживаючих у м. Києві.

```
SELECT  ПІБ
```

```
FROM  СТУДЕНТИ
WHERE  ПІБ Like “А*”
```

Відбираються записи, що містять поле **ПІБ**, для всіх студентів, прізвища яких починаються з букви “А”. Тут в умові добору використовується шаблон *, що заміняє довільне число символів. Застосовуються також такі елементи шаблонів: ? – один будь-який символ; # - одна будь-яка цифра.

7.4. Оператор SELECT. Відбір даних з декількох таблиць

Особливістю запитів на вибірку даних з декількох таблиць є те, що між таблицями, що беруть участь у запиті, повинне встановлюватися з'єднання. З'єднання між таблицями може бути двох видів: внутрішнє і зовнішнє. Розглянемо більш добірно оператор SELECT, що здійснює вибірку даних з таблиць при різних видах з'єднання між ними.

7.4.1. Внутрішнє з'єднання таблиць

Нехай, наприклад, для відбору даних про оцінки студентів записаний такий оператор:

```
SELECT  СТУДЕНТИ.ПІБ, УСПІШНІСТЬ.ОЦІНКА
FROM  СТУДЕНТИ, УСПІШНІСТЬ
WHERE  СТУДЕНТИ.КодСтуд = УСПІШНІСТЬ.КодСтуд
```

Цей оператор установлює внутрішнє з'єднання двох таблиць. Результуючий набір даних при внутрішнім з'єднанні таблиць А і В формується згідно такому правилу:

Зі стовпців (полів), що зазначені в інструкції SELECT, складається проміжний набір даних шляхом зчеплення результуючих стовпців кожного запису з таблиці А и всіх результуючих стовпців з таблиці В;

З отриманого набору даних відкидаються всі записи, що не задовольняють умовам відбору в інструкції WHERE.

Припустимо, що в таблицях СТУДЕНТИ й УСПІШНІСТЬ міститься така інформація:

СТУДЕНТИ (таблиця А)

КодСтуд	ПІБ	Адреса
1	Іванов	…
2	Петров	…

УСПІШНІСТЬ (таблиця В)

КодСтуд	КодПр	Оцінка
1	1	5
1	2	3
2	1	4
2	1	3

Тоді приведений вище оператор SELECT у процесі його виконання формує проміжний і результуючий набори в такий спосіб:

Проміжний НД

С.ПІБ	У.Оцінка
Іванов	5
Іванов	3
Іванов	4
Іванов	3
Петров	5
Петров	3
Петров	4
Петров	3

Результуючий НД

С.ПІБ	У.Оцінка
Іванов	5
Іванов	3
Петров	4
Петров	3

Для скорочення тексту оператора рекомендується використовувати так звані “псевдоніми” таблиць, тобто їхні короткі найменування. Наприклад, з використанням псевдонімів запис попереднього оператора може мати такий вигляд:

SELECT С.**ПІБ**, У.**ОЦІНКА**
FROM СТУДЕНТИ as С, УСПІШНІСТЬ as У
WHERE С.**КодСтуд** = У.**КодСтуд**

У цієї записі оператора в інструкції FROM задані псевдоніми С і У: для таблиці СТУДЕНТИ – псевдонім С, для таблиці УСПІШНІСТЬ – У.

Для створення внутрішнього з'єднання таблиць частіше використовується така форма інструкції FROM:

SELECT С.**ПІБ**, У.**ОЦІНКА**
FROM СТУДЕНТИ as С INNER JOIN УСПІШНІСТЬ as У
ON С.**КодСтуд** = У.**КодСтуд**

Тут службові слова INNER JOIN указують на внутрішнє з'єднання таблиць. За ключовим словом ON вказується умова внутрішнього з'єднання. Поля С.**КодСтуд** і У.**КодСтуд**, які вказані за службовим словом ON, виявляються *ключами зв’язку*, за якими створене з'єднання двох таблиць.

При внутрішньому з'єднанні у результуючу таблицю включаються тільки ті записи, які складаються з існуючих даних обох таблиць.

7.4.2. Зовнішні з'єднання таблиць

Крім розглянутого вище внутрішнього з'єднання існують ще і зовнішні з'єднання. Зовнішнє з'єднання відрізняється від внутрішнього тім, що в результуючий НД включаються *всі* записи стовпців з ведучої таблиці, що поєднуються з можливо порожніми записами з стовпцями другої (відомої) таблиці.

Формат оператора SELECT при зовнішнім з'єднанні такої ж, як і при внутрішньому. Відмінність полягає в тому, що замість слова INNER, що вказує на внутрішнє з'єднання, записується одне з таких слів:

LEFT – ліве зовнішнє з'єднання, при якому ведучої є таблиця, що розташована “ліворуч” від конструкції JOIN;

RIGHT – праве зовнішнє з'єднання, коли ведучої є таблиця, розташована “праворуч”;

FULL – повне зовнішнє з'єднання, коли ведучими є обидві таблиці. У цьому випадку в результуючий НД записі включаються за наступним правилом. Якщо для запису в таблиці А маються записи в таблиці В, що задовольняють умові з'єднання, у результуючий НД включаються всі комбінації з'єднань стовпців з таких записів. У противному випадку в результуючий НД будуть включені стовпці з таблиці А, з'єднані з порожніми стовпцями. Те ж відноситься і до стовпців з таблиці В.

Розглянемо наступний приклад. Нехай є такі таблиці:

СТУДЕНТИ

КодСтуд	**ПІБ**	**Адр**
1	Іванов	…
2	Петров	…
3	Сидоров	…

УСПІШНІСТЬ

КодСтуд	**КодПр**	**Оцінка**
1	1	5
1	2	3
2	1	4
2	1	3
4	3	5

Оператор лівого зовнішнього з'єднання

SELECT **ПІБ**, **Оцінка**
FROM СТУДЕНТИ С LEFT JOIN УСПІШНІСТЬ У
ON С.**КодСтуд** = У.**КодСтуд**

Буде получено такий результуючий НД:

ПІБ	**Оцінка**
Іванов	5
Іванов	3
Петров	4
Петров	3
Сидоров	

Оператор правого зовнішнього з'єднання

SELECT **ПІБ**, **Оцінка**
FROM СТУДЕНТИ С RIGHT JOIN УСПІШНІСТЬ У
ON С.**КодСтуд** = У.**КодСтуд**

створить такий результуючий НД:

ПІБ	**Оцінка**
Іванов	5
Іванов	3
Петров	4
Петров	3
	5

Оператор повного зовнішнього з'єднання таблиць С и У
SELECT **ПІБ**, **Оцінка**
FROM СТУДЕНТИ С FULL JOIN УСПІШНІСТЬ У
ON С.**КодСтуд** = У.**КодСтуд**

створить такий результуючий НД:

ПІБ	**Оцінка**
Іванов	5
Іванов	3
Петров	4
Петров	3
Сидоров	
	5

Контрольні запитання:

1. Як в операторі SELECT створити поле, що обчислюється? Навести приклад.
2. Для чого потрібні групові функції і як вони застосовуються в операторі SELECT? Навести приклади.
3. Що таке “вкладені запити”, для чого вони потрібні? Навести приклад.
4. Як в СУБД *Access* створити запит на відбір даних за допомогою Конструктора?
5. Як задати складні умови відбору при створенні запиту за допомогою Конструктора у СУБД *Access*?

8. Обчислення в запитах. Створення запитів у *Access*

8.1. Обчислення в запитах

В операторі SELECT можна робити обчислення, результати яких будуть міститися в додатковому полі, яке включається у вихідний НД і називається *полем, що обчислюється*. Можливі два способи створення полів, що обчислюються:

- шляхом використання виразу в інструкції SELECT;
- з використанням групових (агрегатних) функцій.

Розглянемо кожний з цих способів.

8.1.1 Створення поля, що обчислюється, за допомогою виразу

Для створення поля, що обчислюється, можна в інструкції SELECT у списку полів записати вираз, наприклад, у такий спосіб:

SELECT **ПІБ**, 2018-**РікНародж**
FROM СТУДЕНТИ

У цьому операторі в результуючому НД поряд з полем **ПІБ** міститься поле, що обчислюється, у якому поміщується вік кожного студента. Створеному полю СУБД привласнить своє (внутрішнє) ім'я, що не буде відбивати змісту інформації, що міститься в ньому. Тому рекомендується створити для поля, що обчислюється, псевдонім, який потім використовується як ім'я поля. У нашому прикладі це можна зробити так:

SELECT **ПІБ**, 2018-**РікНародж** as **Вік**
FROM СТУДЕНТИ

Створюваному полю, що обчислюється, привласнюється ім'я **Вік.**

8.1.2. Застосування групових (агрегатних) функцій

Дуже часто потрібно зробити обчислення деяких підсумкових даних по всьому наборі даних чи по яких-небудь групах даних. У цьому випадку використовуються групові (агрегатні) функції: *Max, Min, Count, Avg, Sum* (див. розділ 8).

Нехай, наприклад, є таблиця СТУДЕНТИ, що має наступну структуру:

СТУДЕНТИ (**КодСтуд**, **КодГр**, **ПІБ**, …).

Підрахувати кількість усіх студентів, про яких зберігаються дані у таблиці СТУДЕНТИ, можна за допомогою наступного оператора:

SELECT *Count*(**ПІБ)** as **Кільк_Студ**
FROM СТУДЕНТИ

Очевидно, що даний запит повертає єдине значення – кількість студентів (поле **Кільк_Студ**), яка у даному випадку збігається з числом записів у таблиці.

Розглянемо більш цікавий приклад – обчислимо середній бал по кожному зі студентів. Нехай, наприклад, інформація про оцінки міститься в таблиці УСПІШНІСТЬ, що має структуру:

УСПІШНІСТЬ (**КодСтуд**, **КодПр**, **Дата**, **Оцінка**).

Текст запиту, що обчислює середній бал для кожного зі студентів, може мати наступний вигляд:

```
SELECT  С.ПІБ,  Avg(У.Оцінка) as Серед_Бал
FROM   СТУДЕНТИ as С LEFT JOIN УСПІШНІСТЬ as У
            ON С.КодСтуд = У.КодСтуд
GROUP BY  С.ПІБ
```

Оскільки тут добір даних утворюється з двох таблиць (поле **ПІБ** береться з таблиці СТУДЕНТИ, а поле **Оцінка**, що бере участь в обчисленнях, - з таблиці УСПІШНІСТЬ), у запиті створюється з'єднання таблиць. Завдяки тому, що створюється ліве зовнішнє з'єднання, у результуючий набір даних будуть включені і прізвища студентів, для яких немає оцінок у таблиці УСПІШНІСТЬ.

В інструкцію GROUP BY повинні включатися всі поля, імена яких зазначені в інструкції SELECT і не беруть участь в агрегатних функціях. У нашому прикладі це поле С.**ПІБ**.

В інструкції SELECT для поля, що обчислюється, одержуваного як результат застосування групової (агрегатної) функції *Avg* до поля **Оцінка**, створюється псевдонім **Серед_Бал**. Цей псевдонім на замовчення буде відображатися в заголовку стовпця середнього бала.

Якщо в запиті, у якому застосовуються групові функції, ще потрібно зробити відбір записів по деякій умові, застосовуваній до підсумкових (групованих) даних, тоді для завдання умови потрібно використовувати інструкцію HAVING (а не WHERE).

Нехай у розглянутому прикладі потрібно в результуючий НД помістити тільки записи, що відносяться до студентів із середнім балом 3,6 і більше. Тоді приведений вище текст запиту потрібно доповнити інструкцією:

HAVING *Avg*(У.**Оцінка**) >= 3,6

8.2. Вкладені запити

Часто буває неможливо вирішити поставлену задачу шляхом використання тільки єдиного запиту. Наприклад, у випадку, коли в умові добору потрібно використовувати заздалегідь невідоме значення, з яким потрібно порівнювати дане значення.

Нехай, наприклад, вирішується задача визначення середнього бала для студента Іванова. Але при цьому в таблиці УСПІШНІСТЬ, у якій зберігаються дані про оцінки, немає поля, що містить прізвища, але є тільки поле **КодСтуд**, яке містить внутрішні коди студентів, що користувачу невідомі. З використанням вкладеного запиту вирішити дану задачу можна в такий спосіб:

```
  SELECT  Avg(Оцінка)
FROM  УСПІШНІСТЬ
GROUP BY КодСтуд
HAVING КодСтуд =
                        (SELECT  КодСтуд
     Вкладений          FROM     СТУДЕНТИ
       запит            WHERE    Фам = "Іванов"
```

Дану задачу можна було б вирішити і без вкладеного запиту, створивши в запиті з'єднання таблиць СТУДЕНТИ й УСПІШНІСТЬ.

Розглянемо приклад задачі, яку не можна вирішити інакше, як з використанням вкладених запитів. Нехай, наприклад, потрібно одержати список студентів, у яких середній бал більше, ніж у студента Іванова. Тому що середній бал Іванова заздалегідь невідомий, для його визначення використовується вкладений запит.

Оператор SELECT, що вирішує цю задачу, виглядає в такий спосіб:

```
SELECT Фам,  Avg(Оцінка)
FROM  СТУДЕНТИ С INNER JOIN УСПІШНІСТЬ У
   ON С.КодСтуд = У.КодСтуд
GROUP BY Фам
HAVING Avg(Оцінка) >
                  (SELECT  Avg(Оцінка)
                   FROM  УСПІШНІСТЬ
     Вкладений    GROUP BY  КодСтуд
       запит       HAVING   КодСтуд =
                                (SELECT  КодСтуд
                    Вклад.       FROM     СТУДЕНТИ
                    запит        WHERE    Фам = "Іванов" ))
```

Тут застосований вкладений запит з двома рівнями вкладеності. Взагалі кількість рівнів вкладених запитів не обмежується.

8.3. Створення запитів у Access

У *Access* маються могутні і зручні інструментальні засоби створення різних запитів. Для створення простих так званих “запитів за зразком” можна використовувати Майстер запитів, що послідовно запитує у користувача потрібну інформацію (найменування таблиць, які будуть використовуватися у запиті; імена полів, які потрібно включити в результуючий НД і інш.) і потім автоматично генерує *SQL*-запит. Створений за допомогою Майстра запит відразу можна виконати і зберегти для подальшої його доробки й удосконалення за допомогою Конструктора запитів.

Для створення більш складних запитів необхідно використовувати Конструктор запитів. Розглянемо більш детально технологію створення запитів в *Access* за допомогою Конструктора запитів. Для створення запиту в режимі Конструктора потрібно виконати наступне:

1) У вікні вашої БД відкрити вкладку “Запити” і натиснути кнопку **Створити**. Відкриється діалогове вікно “Новий запит”, у якому потрібно вибрати спосіб створення запиту. Виберіть спосіб “Конструктор” і натисніть кнопку **ОК**.

2) Відкриється вікно Конструктора запитів і відразу ж на його фоні відкриється діалогове вікно “Додавання таблиці”. Виберіть кнопкою миші потрібні для запиту таблиці і натисніть кнопку **Додати**. Після цього закрийте вікно “Додавання таблиці” кнопкою **Закрити.**

3) Вікно Конструктора запитів має вид, показаний на рис. 8.1. Вікно розділене на 2 панелі: верхню і нижню. Верхня панель називається схемою даних запиту, на якій розміщені списки полів таблиць, що були додані в запит. У списках полів верхній рядок відзначений зірочкою *, що позначає всі поля. *Access* автоматично встановлює зв'язок між полями з однаковими іменами і типом даних.

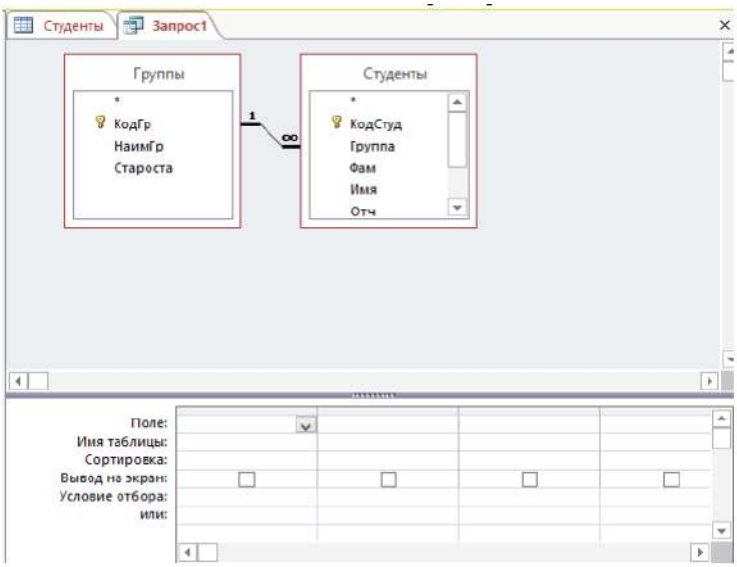


Рисунок 8.1 - Вид вікна Конструктора запитів

Нижня панель називається бланком запиту, що заповнюється в такий спосіб:

- у рядок “Поле” помістити імена полів, що повинні бути включені в запит. Зробити це можна, наприклад, способом перетаскування потрібного поля зі Схеми даних у рядок “Поле”;
- у рядку “Сортування” вибрати порядок сортування для поля, по якому потрібно сортування;
- у рядку “Вивід на екран” відзначити поля, що повинні бути виведені в результуючу таблицю;
- у рядках “Умови відбору” і “чи” задати умови відбору записів.

Умови відбору даних (записів) задаються виразами, що можуть складатися з операндів і операторів (операцій) порівняння. У якості операндів можуть використовуватися:

- літерали: “Іванов”, “СОБД”, “А*”, #1.09.99# ;
- константи: *True*, *False*, *Null* ;
- ідентифікатори – посилання на поля: [Ім'я таблиці] .[Ім'я поля].

У виразах можуть застосовуватися наступні операції і предікати:

= , < , > , <> , And , Or , Not , Between , In , Like.

4) Зачинити вікно Конструктора запитів. При цьому відкривається діалогове вікно, у якому потрібно увести ім'я запиту, що був створюваний. Вид діалогового вікна зображено на рис. 8.2. Після введення потрібного імені натисніть кнопку **ОК.**

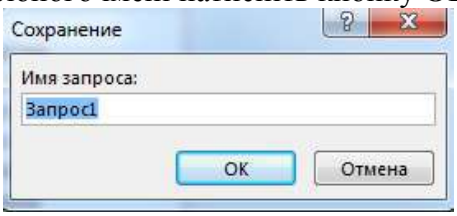


Рисунок 8.2 - Вікно для введення імені запита

Створений запит можна запустити на виконання подвійним щигликом миші на значку запиту у вікні БД чи натисканням кнопки **Відкрити.**

У режимі Конструктора запит можна запустити на виконання командою меню **Конструктор**|**Виконати** натиснув кнопку 

на панелі інструментів.

Контрольні запитання:

1. Отримати список прізвищ студентів, які мають оцінку "5". У вихідний НД включити стовпці: НазваГр, Прізвище.
2. Отримати список студентів групи "САД-21", що мають середній бал більше 4,0. У вихідний НД включити стовпці: Прізвище, СередБал.
3. Отримати список студентів групи "САД-21", що мають оцінки "5" з ОБДЗ. У вихідний НД включити стовпці: Прізвище, Ім'я, По-батькові.
4. Отримати середні бали по предметам для студента Іванова І.І. У вихідний НД включити стовпці: Предмет, СередБал.
5. Отримати середні бали і прізвища всіх студентів, які мають оцінки з фізики. У вихідний НД включити стовпці: Прізвище, СередБал.
6. Визначити кількість оцінок "2" в кожній групі. У вихідний НД включити стовпці: Група, Число_2.
7. Визначити середні бали по групах. У вихідний НД включити стовпці: Група, СередБал.
8. Визначити середні бали і кількість оцінок з предметів. У вихідний НД включити стовпці: Предмет, КількОцінок, СередБал.
9. Визначити кількість оцінок з фізики у студента Іванова І.І. У вихідний НД включити стовпці: Прізвище, Предмет, СередБал.
10. Визначити кількість оцінок у студента Іванова І.І. за видами занять. У вихідний НД включити стовпці: Прізвище, ВідЗанят, СередБал.

9. Створення форм у *Access*

9.1. Види та призначення форм

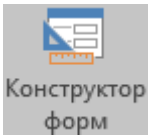
Форма є одним із об'єктів СУБД *Access*. Вона забезпечує найбільше гнучкий спосіб для введення, передогляду та корегуванню взаємопов'язаних даних в таблицях и запитах створеної бази та також для розробки панелей керування базою в вигляді, зручному для користувача. Працюючи з формою, користувач може змінювати значення у полях, виконувати розрахунки загальних функцій та складних виразів, контролювати дані, що додаються, установлювати обмеження до доступу, відкривати таблиці , інші форми та звіті графічними засобами, як командні кнопки, вкладки, кнопки-списки. Ці графічні об'єкті мають назву - елементи керування.

Основні елементи керування у формі: поле, напис, група, перемикач, позначка, вимикач, поле зі списком, список, кнопка, рисунок, приєднана рамка об'єкта, вільна рамка об'єкта, набір вкладок, підпорядкована форма, розрив сторінки, лінія, прямокутник, додаткові елементи *ActiveX*.

Графічний елемент керування у форми може забезпечити:

- перегляд записів і їх оброблення (додання, вилучення, друкування, відновлення);
- роботу з формою та зі звітами (відкривання, перегляд, закривання, друкування);
- роботу з додатками (запускання додатка, якій був виконаний у Word, Excel);
- відкривати OLE- об'єкти (рисунки, графіки, фото).

Конструкцію форми визначає користувач. При цьому він вказує, які графічні елементи керування мають бути включеними у форму, з яких таблиць та запитів, зміст яких полів треба відображати.

Створити форму можливо декількома способами. Для конструювання нової форми у вікні бази даних необхідно вибрати вкладку **Створити** та клацнути мишой по кнопці **Конструктор форм,** 

на екрані з'явиться діалогове вікно «Нова форма», в якому вказані способи створення (рис.9.1).

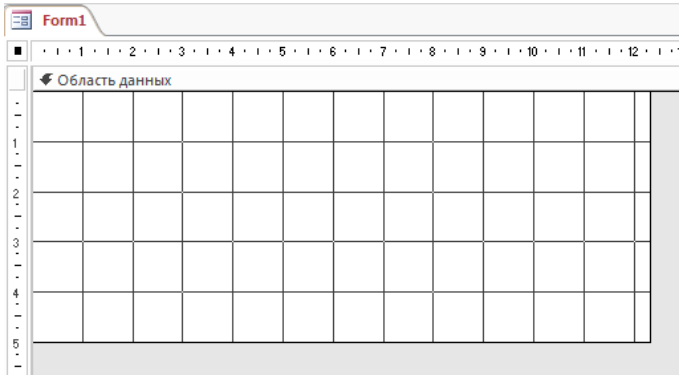


Рисунок 9.1 - Діалогове вікно «Нова форма»,

За допомогою **Конструктора** створюється порожня форма, в яку далі додають поля таблиці, елементи керування, редагують положення, розмір, колір шрифту, фон.

За допомогою **Майстра** автоматично створюють вид форми, вибирав шаблон із чотирьох пропонованих видів форм представлення даних, якій потім можливо корегувати в режимі Конструктора: "в столбец", "ленточная", "табличная" та "выровненная".

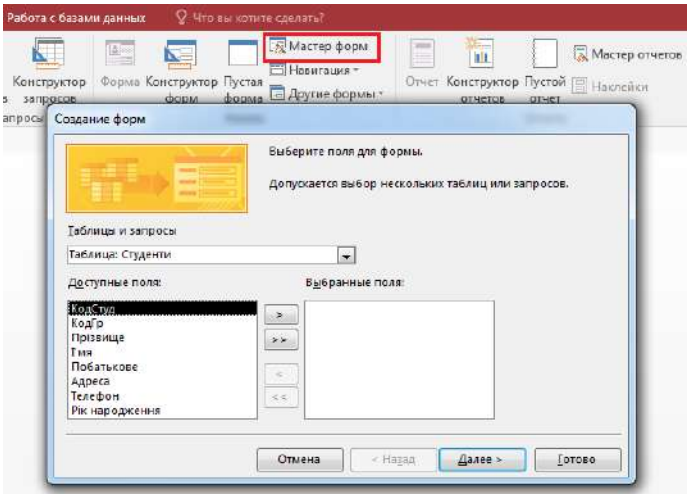


Рисунок 9.2 - Діалогове вікно “Нова форма”

У режимі **Форма** створюється форма практично без участі користувача, яку потім можна корегувати Конструктором.

Наприклад, форма Студенти, що створена у режимі “ленточного” виводу полів, має вид, зображений на рис.9.3.

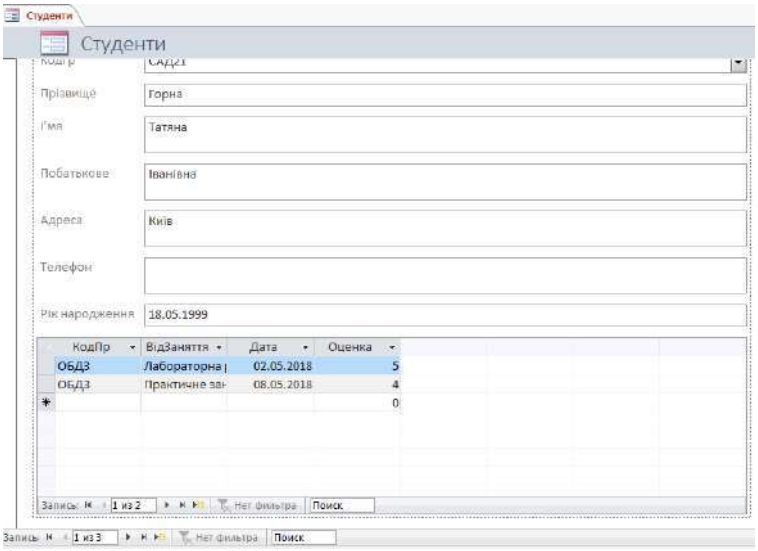


Рисунок 9.3 - Форма “ленточного” типу

Режим **Навігація** забезпечує подання даних у різному графічному вигляді (рис.9.4).

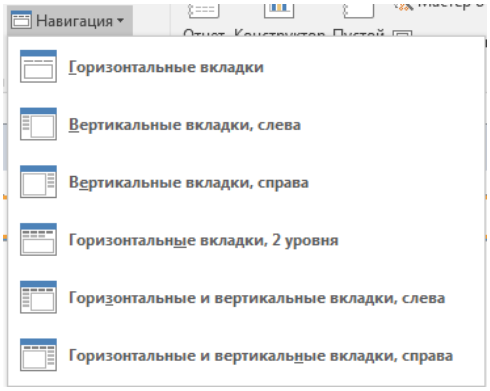
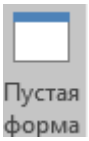


Рисунок 9.4 - Меню режиму **Навігація**

Режим **Пуста форма** 

створює форму без елементів управління та формату.

9.2. Створення форм за допомогою Конструктора

Для створення форми в цьому режимі необхідно в діалоговому вікні вибрати пункт «Конструктор», а у якості джерела даних для неї – таблицю або запит. Відчинитися діалогове вікно «Форма1: форма» (рис. 9. 5).

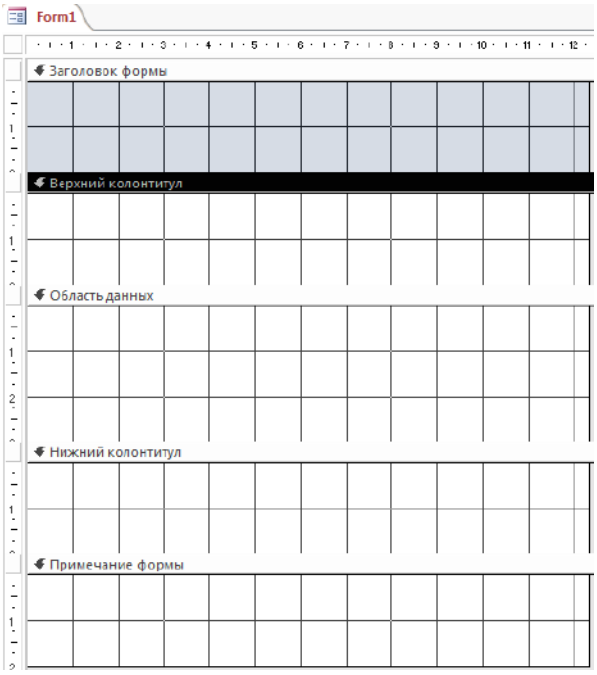


Рисунок 9.5 - Форма у режиму **Конструктор**

Робочим полем Конструктора є форма, що складається з таких ділянок: «Заголовок формы», «Верхний колонтитул», «Область данных», «Нижний колонтитул» та «Примечание формы», які додаються з контекстного меню (рис.9.6).

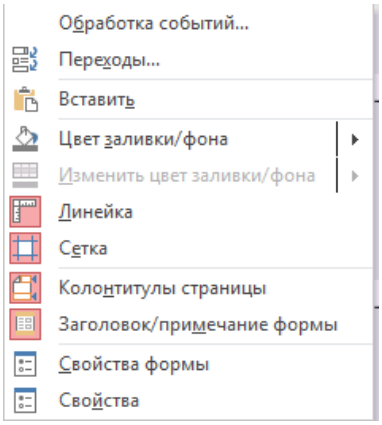


Рисунок 9.6 - Контекстного меню форми

Для розміщення на них потрібних об'єктів (полів, тексту, графічних об'єктів) необхідно використати елементи трьох панелей інструментів – «Конструктор форм», «Панель елементів» і «Формат» (рис.9.7, 9.8).

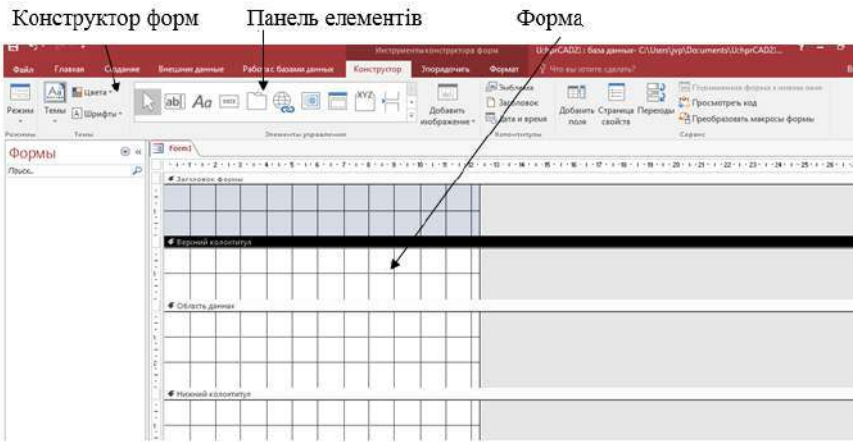


Рисунок 9.7. Діалогове вікно «Форма1: форма»

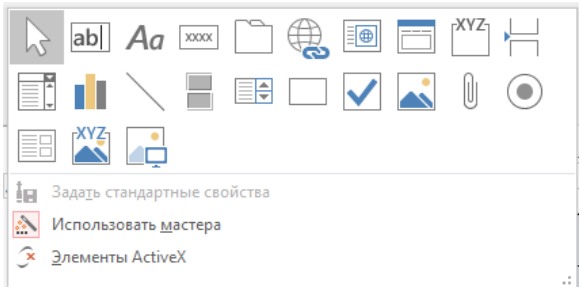


Рисунок 9.8. Панель інструментів «Конструктор форм»

Призначення кнопок панелі інструментів «Конструктор форм» пояснюють довідки.

Для конструювання табличної форми, у яку буде додана інформація з одної таблиці, необхідно клацнути мишею на кнопці **Список полів** панелі інструментів «Конструктор форм». На екрані з'явиться вікно зі списком полів вибраної таблиці або запиту, наприклад, таблиці СТУДЕНТИ (рис.9.9).

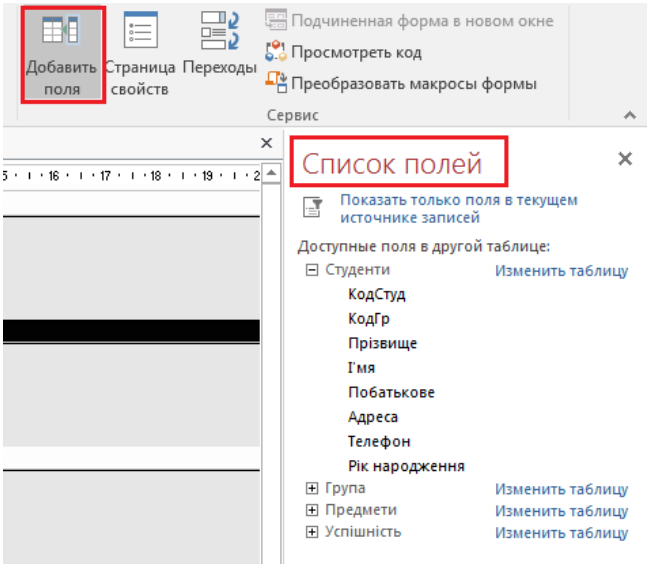


Рисунок 9.9 - Список полів

Потрібні поля необхідно скопіювати із таблиці в «Область данных» форми (рис.9. 10).

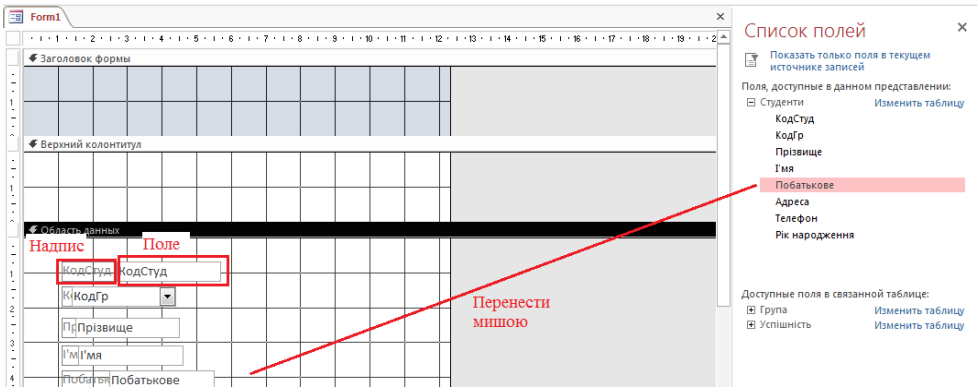


Рисунок 9.10 - Встановлення полів у форму

Розділи форми можливо відформовувати (призначать ім'я , встановити розмір полів, фон та інше), заповнить даними. Для цього клацанням на ньому правою кнопкою миші потрібно викликати контекстно залежне меню, у якому вибрати команду **Свойства** і в діалоговому вікні, що з'явиться, установити необхідні елементи та властивості (рис.9.11).

Завершальною операцією конструювання форми є її збереження в базі даних. Після цього треба закрити вікно «Конструктор форм» і ввести ім'я форми.

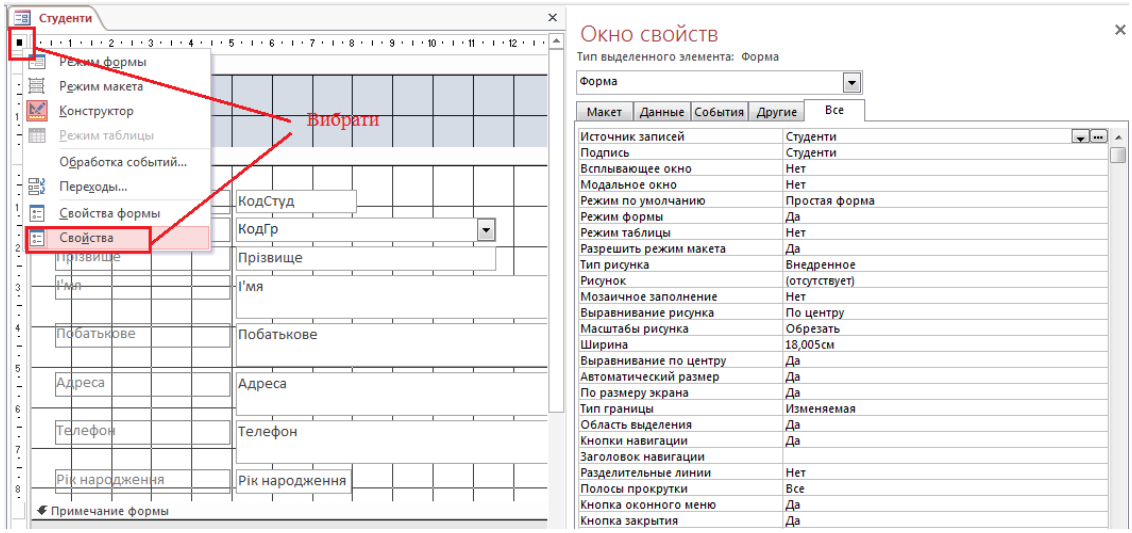


Рисунок 9.11 - Редагування властивостей розділів форми

Наприклад, щоб переглянути список студентів групи, скорегувати дані, що їх стосуються, увести нові записи або вилучити зайві, необхідно відкрити базу даних, клацнувши мишею на вкладці «Формы» та відкрити форму «Студенти». Внаслідок цього на екрані з'явиться форма.

За допомогою форми, що була створена, можна працювати з таблицею СТУДЕНТИ, тобто переглядати записи, вводити при потребі зміни, доповнювати таблицю новими записами.

9.3. Створення форм за допомогою Майстра

Для оперативного конструювання форми, а також створення багатотабличної форми доцільно використати **Майстер форм.** У цьому режимі система створює форму поетапно – з урахуванням відповідей користувача на її запитання

Для створення нової форми необхідно в діалоговому вікні «Новая форма» клацнути мишею на рядку «Майстер форм», а зі списку таблиць і запитів (якщо створюється багатотаблична форма) вибрати те поле, яке використовуватиметься як первинне джерело даних. На екрані з'явиться діалогове вікно **Майстра форм.**

Воно дає змогу відібрати потрібні таблиці (запроси) яки були створенні у БД, імена полів із цих таблиць та включить їх в форму. Для цього потрібно додати їх у вікні «Доступні поля» (рис.9.12).

В другому діалоговому вікні, склад якого залежить від кількості таблиць та запитів, відібраних на першому етапі, потрібно указати головну таблицю, з якої буде братися дані. Можливо вибрати один із варіантів стрівання формі – «Підпорядкована форма» або «Зв'язана форма».

У першому випадку створюється форма, яка міститиме один запис головної таблиці і відповідні записи з підпорядкованої таблиці (рис.9.13).

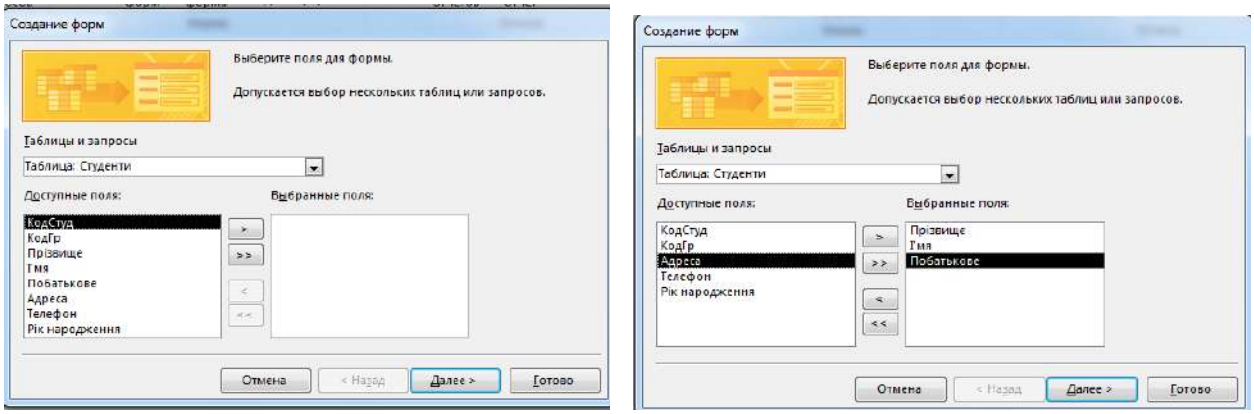


Рисунок 9.12 - Діалогове вікно **Майстра форм**

У полі **Таблиці та запити** можна вибрати таблицю з якою додається підпорядкована інформація (рис.9.13) та спосіб взаємодії форм (Підпорядкована або звязана).

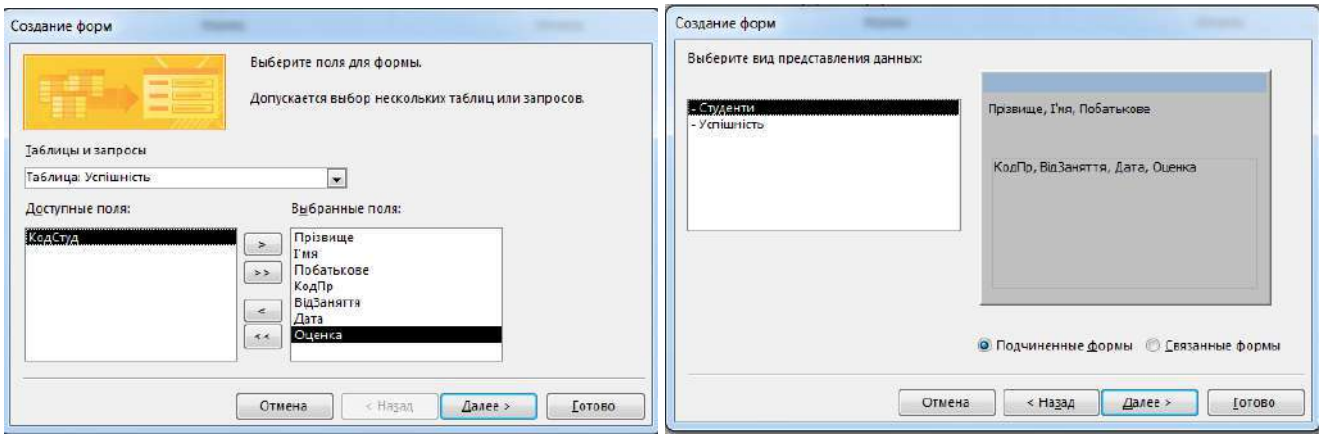


Рисунок 9.13 - Редагування підпорядкованої форми

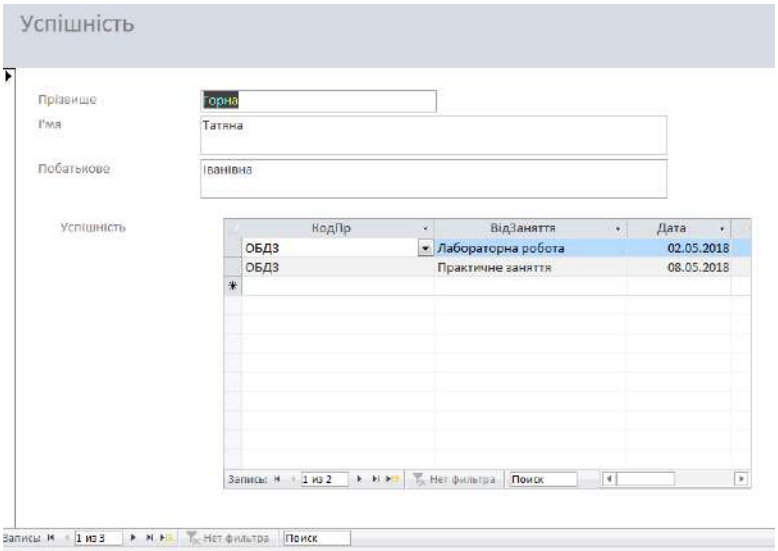


Рисунок 9.14 - Підпорядкована форма

У другому варіанту – додаткова форма, пов'язана з головною формою (рис.9.15).

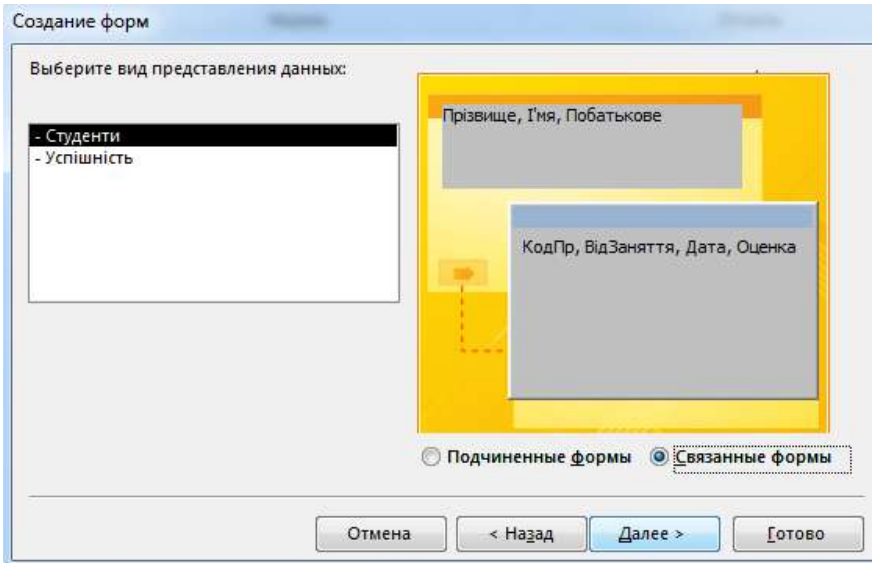


Рисунок 9.15 - Вибір пов’язаної форми

Для більшої яскравості можливо вибрати один із 10 стилів оформлення форми.

На останньому етапі формуються заголовки головної та пов'язаної таблиць. При натисканні на кнопку **Успішність** у головної форми з’явиться пов’язана форма (рис.9.16).

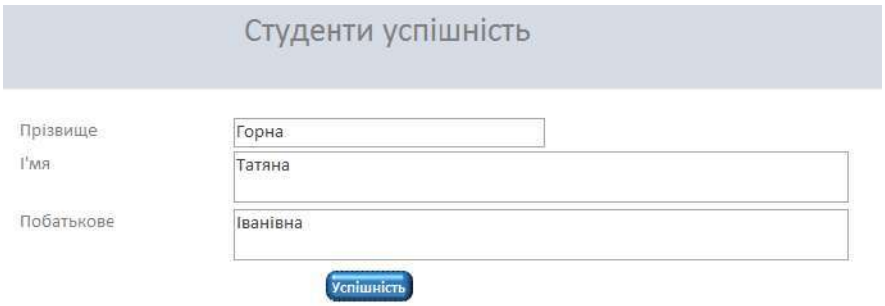


Рисунок 9.16 - Пов’язана форма

Якщо вибрано одну таблицю, то на другому етапі на екрані з'являється діалогове вікно, за допомогою якого можна вибрати один із варіантів розміщення вікон до виведення (введення) даних: «в один стовпчик», «стрічкова», «таблична» або «вирівняна». Після завершення всіх етапів конструювання в форму можливо включити за допомогою **Конструктора** різні елементи керування, які полегшають пошук та керування даними.

9.4. Додання у форму елементів керування

Усі елементи керування можуть бути пов'язаними, непов'язаними або такми, що визначаються.

Пов'язаний елемент керування приєднується до поля базової таблиці. Використовують його для відображення, введення або відновлення значень полів таблиць бази даних.

Вільний елемент керування джерела даних не має. Такі елементи застосовують для виведення на екран даних, ліній, прямокутників або рисунків.

Елемент керування, що визначається, призначений для виконання розрахунків. Як джерело даних у ньому використовується вираз, у якому можуть бути дані полів базової таблиці, а також дані іншого елемента керування форми.

У разі додавання у форму елементів керування, пов'язаних із полями даних таблиць, необхідно в режимі «Конструктор» відкрити список полів (команда **Вид**|**Список полів)** та вставити вибране поле на потрібне місце листа конструктора форми.

Якщо виникне потреба додати у форму одночасно кілька суміжних або несуміжних полів, то треба виділити їх при затисненій клавіші <***Ctrl***>, відповідно, та вставити на потрібне місце.

У вікні «Конструктор форм» можна також створити поле, що обчислюється, для яких небудь розрахунків, наприклад, для отримання середнього балу студента.

Додавання у форму непов'язаних елементів керування здійснюється за допомогою панелі керування «Панель елементів». Призначення кнопок панелі елементів керування наступне:


– вибір елемента, розділу чи форми;


– вмикання і вимикання Майстра створення елементів керування;


– додання напису, заголовка або інструкції;


– створення полів уведення, що відображають вміст полів даних;


– додання групи перемикачів або вимикачів;


– додання вимикача, що працює за логікою «так – ні»;


– додання перемикача для вибору одного значення із багатьох;


– додання позначки, що працює за логікою «так–ні»;


– створення комбінованого списку;


– створення звичайного списку;


– додання кнопки для виконання команд;


– вставлення рисунка;


– вставлення рамки, що містить об'єкт OLE;


– вставлення рамки, яка містить посилання на об'єкт OLE;


– вставлення розриву сторінки;


– вставлення форми, що складається з кількох вкладок;


– додання підпорядкованої форми або підпорядкованого звіту;


– рисування прямих ліній;


– рисування прямокутника або квадрата;

Для додання елементів необхідно клацнути на кнопці потрібного елемента, розміщеного на панелі інструментів, а потім - на тому місці, де має бути розташований верхній лівий кут цього елемента, потім потрібно вказати дії, які буде виконувати елемент. Наприклад, на рис.9.15 кнопка **Успішність** відкриває форму **Успішність.**

До сконструйованої форми може виникнути потреба ввести вираз для визначення даних, яких немає в таблиці або запиті (наприклад, поле).

При використанні елементів керування, що визначаються, розрахунки проводять для кожного переходу до нового запису. Такі вирази стосуються полів елементів керування, які мають властивість джерела запису і можуть вводитися вручну або за допомогою вікна **Будівник виразів** (рис. 9.17).

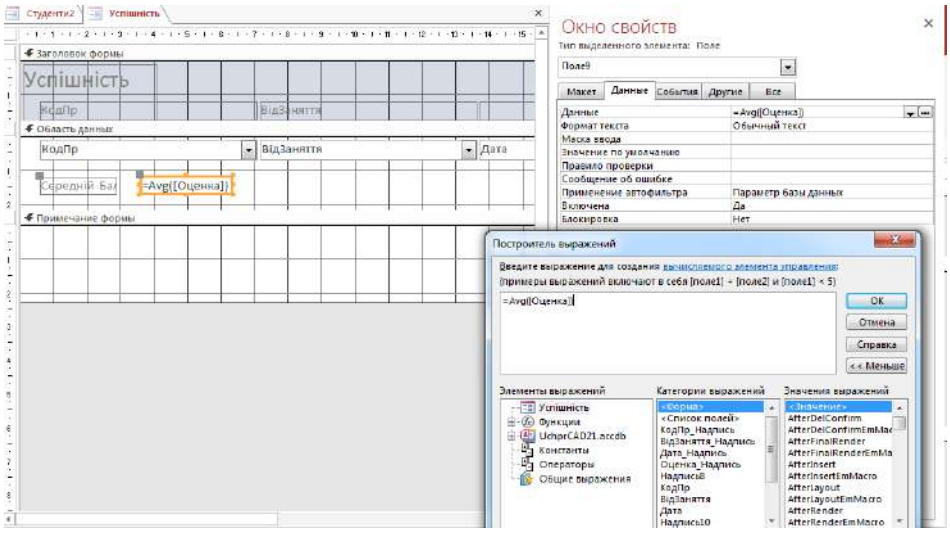


Рисунок 9.17 - Вікно **Побудувача виразів**

Для введення виразу вручну в режимі «Конструктор» необхідно на ділянку даних за допомогою панелі інструментів помістити поле введення, клацнути мишею на ньому і ввести вираз, починаючи зі знаку рівності.

Наприклад, для обчислення середнього балу студента у формі УСПІШНІСТЬ (рис.9.17) додане нове поле з надписом **Середній Бал** у якому виконується у властивісті **Дані** вираз := *Avg*([**Оцінка**]). Для введення виразу за допомогою **Побудувача виразів** треба після додання поля введення викликати за допомогою правої клавіші миші контекстнозалежне меню, активізувати команду **Властивості,** відкрити вкладку «Дані» у вікні властивостей і клацнути мишею на кнопці **Побудувати...** у рядку «Дані», як показано на рис.9.17.

Форма, з розрахунковим полем **Ср_балл**, має вигляд, що наведена на рис.9.18.

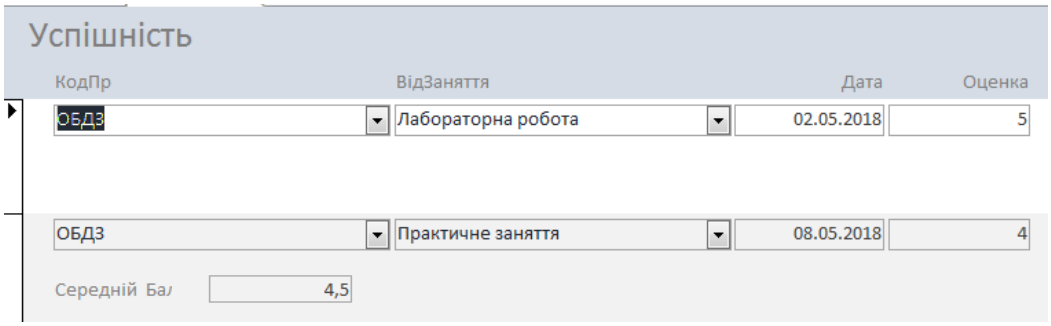


Рисунок 9.18 - Форма з розрахунковим полем

Аналогічно у форму додають елементи керування для логічних полів у вигляді вимикачів , позначок, розміщені списки і поля зі списками , кнопки, рисунки.

Використовуючи форми, можливо редагувати записи в таблицях , установив обмеження на поля , які потрібно захистити від випадкового корегування.

Для обмеження дозволу необхідно відкрити форму у режимі **Конструктор,** викликати контекстне меню, у якому вибрати команду **Свойства** (рис.9.19)

В закладці **Все** встановити в рядку **Включена** команду **Нет**, що зробить неактивним поле **Оцінка** при відкритої формі (курсор неможливо встановити на дане поле) . В рядку **Блокировка** команда **Да** робіт неможливим корегувати запис у поли з клавіатури.

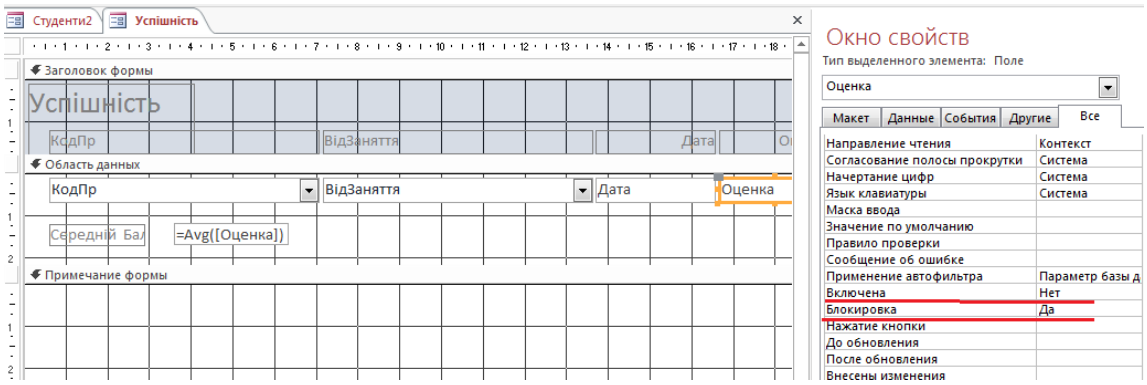


Рисунок 9.19. Вставлення можливості **Доступ** та **Блокіровка**

Контрольні запитання:

1. Призначення та види форм.
2. Методика створення підпорядкованої форми.
3. Призначення кнопок панелі елементів форми.
4. Методика створення розрахункових полів.
5. Як виконати захист форми?

10. Об'єктно-орієнтоване проектування БД у *Delphi*

10.1. Delphi – призначення, загальні відомості

Додаток баз даних, як випливає вже з його назви, призначено для взаємодії з деяким джерелом даних — базою даних (БД). Взаємодія має на увазі одержання даних, їхнє представлення у визначеному форматі для перегляду користувачем, редагування відповідно до реалізованого в програмі бізнес- алгоритмами і повернення оброблених даних назад у базу даних.

Як джерело даних можуть виступати як власне бази даних, так і звичайні файли — текстові, електронні таблиці і ін. Ми будемо розглядати додатка, що працюють з базами даних.

Бази даних обслуговуються спеціальними програмами — системами управління базами даних (СУБД), що поділяються на локальні, переважно однокористувальницькі, призначені для настільних додатків, і серверні — мережні, багатокористувальницькі, що функціонують на виділених комп'ютерах — серверах. Головний критерій такої класифікації — обсяг бази даних і середнє навантаження на СУБД. Проте, незважаючи на розмаїтість реалізацій, загальна архітектура додатка баз даних залишається незмінної.

Сам додаток включає механізм одержання і відправлення даних, механізм внутрішнього представлення даних у тім або іншому виді, користувальницький інтерфейс для відображення і редагування даних, бізнес-логікі для обробки даних.

Механізм одержання і відправлення даних забезпечує з'єднання з джерелом даних. Він повинний "знати", куди йому звертатися і який протокол обміну використовувати для забезпечення двунаправленного потоку даних.

Механізм внутрішнього представлення даних є ядром додатка баз даних. Він забезпечує збереження отриманих даних у додатку і надає них по запиті інших частин додатка.

Користувальницький інтерфейс забезпечує перегляд і редагування даних, а також керування даними і додатком у цілому.

Між додатком і базою даних знаходиться спеціальне програмне забезпечення (ПО), що зв'язує програму і джерело даних і керуючим процесом обміну даними. Це ПО може бути реалізовано найрізноманітнішими способами, у залежності від обсягу бази даних, розв'язуваних системою задач, числа користувачів, способами з'єднання додатка і бази даних. Проміжне ПО може бути реалізоване як оточення додатка, без якого воно взагалі не буде працювати, як набір драйверів і динамічних бібліотек, до яких звертається додаток, може бути інтегроване в сам додаток. Нарешті, це може бути окремий вилучений сервер, що обслуговує тисячі додатків.

Джерело даних являє собою сховище даних (саму базу даних) і СУБД, що керує даними, що забезпечують цілісність і несуперечність даних. Ми докладно зупинимося на способах розробки додатків баз даних у Delphi.

Delphi – одна з найбільше широко розповсюджених сучасних систем програмування. Дозволяє створювати як прості, так і складні професійні комплекси програм, призначені для роботи на платформі ОС *Windows*. Уже розроблений аналог *Delphi*, що одержав назву *Kylix*, призначений для роботи на платформі ОС *Linux*.

Delphi увібрала в себе усі відомі сучасні технології програмування. По-перше, у *Delphi* повною мірою реалізована технологія *об'єктно-орієнтованого програмування* на базі мови *Object Pascal* (зараз його вже називають мовою *Delphi*). По-друге, у *Delphi* доведена до досконалості технологія *візуального проектування* програм, завдяки чому в середовищі *Delphi* можна дуже швидко створювати досить складні сучасні прикладні програми. Нарешті, середовище *Delphi* настільки зручне, що тривалий і кропіткий процес розробки програм стає навіть приємним і комфортним.

Однак, щоб скористатися всіма можливостями, наданими *Delphi*, і відчути їхні переваги, необхідно знати і розуміти базові принципи, реалізовані в *Delphi*. Хоча раніше ви вже знайомилися з ними, ще раз розглянемо коротко основні положення і поняття об'єктно-орієнтованого програмування (ООП) і особливості реалізації в *Delphi* принципів візуального проектування програм.

У Delphi реалізоване досить велике число різноманітних технологій доступу до даних. Але послідовність операцій при конструюванні додатків баз даних залишається майже однакової та в роботі використовуються по суті ті самі компоненти, дороблені для застосування з тією або іншою технологією доступу до даних.

Існують загальні підходи до розробки додатків баз даних у Delphi, базові класи і механізми, що не зміняться, чи виберіть ви для вашого додатка Borland Database Engine (BDE), Microsoft Active Data Objects (ADO) або dbExpress.

10.2. Об'єктно-орієнтоване програмування. Базові класи

ООП – це в даний час природний і загальноприйнятий підхід до проектування програм. Наочним прикладом об'єктно-орієнтованого підходу може служити графічна оболонка *Windows*. Коли Ви відкриваєте будь-як програму в ОС *Windows*, Ви бачите вікна, меню, кнопки, списки і т.п. Усе це об'єкти. Причому, самі по собі вони нічого не роблять, але вони "знають", що робити і як "поводитися" у тім чи іншому випадку. Наприклад, кнопка "знає", що потрібно зробити при клацанні на цій кнопці. Вікно "знає", що потрібно перемалювати себе з новими розмірами, якщо користувач робить протягання однієї з границь вікна. Можна привести ще багато подібних прикладів. Усе це приклади зовнішнього прояву поводження об'єктів. Але що ж таке об'єкт?

Об'єкт можна визначити як структуру даних, у якій *об'єднані дані* і *процедури* (*функції*), застосовувані до цих даних. В ООП уведений спеціальний тип даних – **клас**, у якому визначаються дані і процедури (функції), називані методами. *Екземпляри цього типу* даних є **об'єктами.**

Розглянемо простий фрагмент програми на мові *Pascal*, у якому створюється об'єкт.

```
Type
TObj =  class(TObject)
A: Integer;
B: Real;          } оголошення даних (полів, властивостей)
…….....
procedure P1;
function P2: Real; } оголошення методів (процедур і функцій)
……................
end;
Var
Obj  :  TObj;     } оголошення змінної об'єктного типу
……..
begin
Obj:= TObj.Create;   створення екземпляра класу (об'єкта)
……................
Obj.P1;
if Obj.P2>B then …...   } застосування методів P1 і P2
……..............
end.
```

Оголошені в класі дані можуть бути просто змінними того чи іншого типу – їх називають полями, чи властивостями.

Поля використовуються для внутрішніх потреб у класі і звичайно ззовні не доступні. У приведеному вище прикладі A і B – це поля.

Властивості – це такі ж змінні, але до них можливий доступ ззовні об'єкта. Властивості є характеристиками об'єкта, що визначають його індивідуальні особливості і поводження об'єкта. Через властивості об'єкта здійснюється взаємодія об'єкта з зовнішнім середовищем.

У компонентах *Delphi* (це теж об'єкти) маються спеціальні властивості, що називають **подіями**. Це властивості процедурного типу, призначені для програмування реакцій на ті чи інші повідомлення *Windows*.

Наприклад, кнопка як об'єкт Delphi має безліч властивостей-подій, при настанні яких повинні виконуватися ті чи інші дії. Так, при натисканні кнопки (при клацанні на ній мишею) настає подія *OnClick*. В оброблювачі цієї події можна запрограмувати потрібні дії.

Сполучення в об'єкті даних і процедур маніпулювання цими даними називають *принципом інкапсуляції*.

Іншим найважливішим принципом, що лежить в основі ООП, є *принцип спадкування*. Суть принципу спадкування полягає в тому, що класи можна створювати (породжувати) шляхом спадкування від вже існуючих класів. Породжений клас називають *класом-нащадком*, а клас, від якого він був породжений – *класом-предком*. У класі-нащадку стають доступними дані і методи, що оголошені в класі-предку. При необхідності, у класі-нащадку можна увести свої нові властивості і методи і тим самим розширити можливості класу-нащадка, зробити їх більш багатими в порівнянні з можливостями класа-предка.

У *Delphi* створена і використовується розвита ієрархія базових класів, у яких реалізовані різноманітні функції, практично будь-які, котрі можуть знадобитися у програмах, що розроблюються. Фрагмент ієрархії найбільш важливих (базових) класів, призначення яких повинний розуміти будь-який розроблювач програм *Delphi*, зображений на рис. 10.1.

Клас **TObject** є родоначальником всієї ієрархії класів *Delphi*. У ньому реалізовані функції, якими повинний володіти будь-який об'єкт. Такі функції дві: створення екземпляра класу (метод Create) і його знищення (мтод Destroy). Будь-який об'єкт повинний "уміти" виконувати ці операції.

Створення об'єкта відбувається в результаті виклику методу Create, який називають *конструктором* . При створенні об'єкта для нього в оперативній пам'яті виділяється область адресного простору, створюється покажчик (посилання) на цю область пам'яти Покажчик на екземпляр об'єкта передається в змінну об'єктного типу, що надалі є ідентифікатором об'єкта в програмному коді.

У приведеному вище прикладі об'єкт з ім'ям Obj створюється таким викликом конструктора: Obj:=TObj.Create; , установлюються початкові значення полів і властивостей об'єкта. Тут нагадаємо, що ім'я будь-якого класу починається з літери "T" для того, щоб підкреслити, що це *ім'я типу даних*. Ім'я об'єкту, як і ім'я будь-якої змінної, визначає користувач згідно діючим правилам і обмеженням.

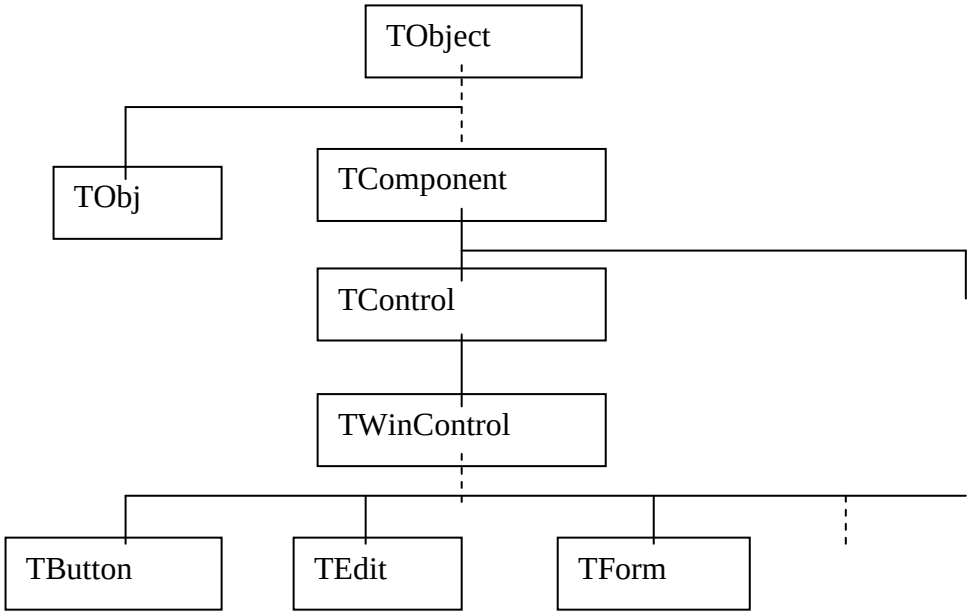


Рис. 10.1 - Фрагмент ієрархії базових класів *Delphi*. Компоненти для роботи з БД.

Для знищення об'єкта призначений метод Destroy, що називається *деструктором*. При виклику деструктора відбувається звільнення пам'яті, що була зайнята об'єктом.

Клас **TComponent** є родоначальником усіх компонентів *Delphi*. Компоненти (це теж об'єкти) є тими "цеглинками", з яких збирається будь-яка програма *Delphi* (це форми, меню, кнопки і т.д.). Від звичайних об'єктів компоненти відрізняються тим, що вони мають такі властивості, що дозволяють їх використовувати і настроювати у візуальному середовищі розробки *Delphi*.

Зокрема, від класу **TComponent** породжені численні класи компонентів, признаених для роботи з БД. З ними пізніше ви познайомитеся більш докладно.

Клас **TControl** є базовим для усіх *візуальних компонентів*. У ньому уводяться властивості і методи, що забезпечують їхнє відображення на екрані.

Клас **TWinControl** додає до властивостей і методів **TControl** властивості і методи, якими повинні володіти будь-які віконні елементи. Віконні елементи керування (теж об'єкти) повинні дозволяти виконувати такі операції:

- одержувати і передавати фокус керування під час роботи програми;
- сприймати керуючі події від миші і клавіатури;
- "уміти" розміщати на собі інші елементи керування.

Віконними елементами керування є не тільки форми, але і практично всі стандартні елементи керування *Windows* – списки, кнопки, редактори і т.д.

10.3. Візуальне проектування програм у Delphi

Візуальне проектування програм – це така технологія, коли програма як би "збирається" з елементів, якими є компоненти *Delphi*. Причому, ця "зборка" виробляється на екрані візуально у вікні графічного редактора.

Головна цінність візуального проектування полягає в тому, що у результаті цього візуального проектування автоматично генерується код програми (текст мовою *Pascal*). Завдяки цьому в *Delphi* виявляється можливим створити досить складну (і корисну) програму не написавши жодного оператора мовою *Pascal*.

Тепер більш докладно розглянемо особливості інтегрованого середовища розробки *Delphi*, у якій і виробляється візуальне проектування програм. При запуску (завантаженні) *Delphi* спочатку відкриваються чотири вікна (рис. 10.2):

- Головне вікно (Delphi 7 – Project1);
- Вікно Інспектора Об'єктів (Object Inspector);
- Вікно Конструктора форм (Form1);
- Вікно Редактора коду (Unit1.pas).

У головному вікні маються такі елементи: головне меню, панелі інструментів, палітра компонентів.

Палітра компонентів є свого роду "вітриною" найбагатшої бібліотеки компонентів *Delphi*. Усі наявні в бібліотеці компоненти згруповані в окремих сторінках (вкладках).

У Репозитории Delphi відсутній окремий шаблон для додатка баз даних. Тому, як і будь-який інший додаток Delphi, додаток баз даних починається зі звичайної форми. Інтерфейс додатка створюється з використанням стандартних і спеціалізованих візуальних компонентів на звичайних формах. Візуальні компоненти відображення даних розташовані на сторінці Data Controls Палітри компонентів. У більшості вони являють собою модифікації стандартних елементів керування, пристосованих для роботи з набором даних.

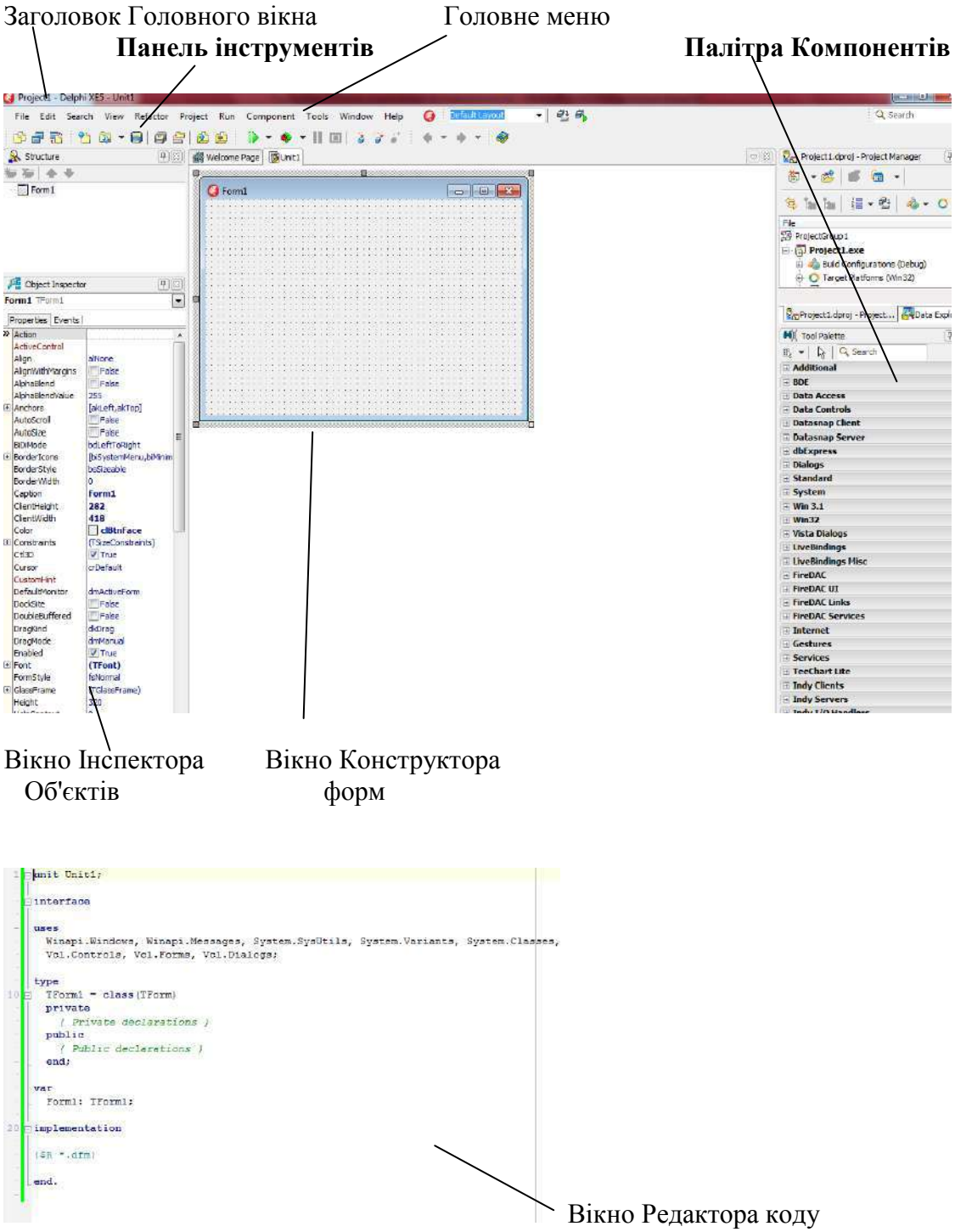


Рисунок 10.2 - Вид екрана в момент після завантаження середовища *Delphi*

Вікно Конструктора форм спочатку знаходиться в центрі екрана і має заголовок Form1. Вікно по суті є графічним редактором, у якому здійснюється візуальне проектування прикладної програми шляхом розміщення на формі компонентів з Палітри Компонентів. Призначення деяких сторінок Палітри компонентів наступне:

Standard – стандартні, найчастіше використовувані компоненти (кнопки, списки, мітки, редактори і т.д.);

Additional – додаткові компоненти;

Win32 – компоненти 32-розрядного інтерфейсу *Windows*;

System – компоненти доступу до системних функцій;
Data Access – компоненти доступу до БД;
Data Controls – елементи керування, для роботи з БД;

У Палітрі Компонентів *Delphi 7* усього мається 19 сторінок. Палітру можна набудовувати: додавати нові компоненти, видаляти наявні.

Для розміщення на формі компонента потрібно виконати такі дві операції:

1) вибрати в Палітрі Компонентів потрібний компонент і помістити його в потрібне місце на формі. На формі з'явиться зображення компонента (рис. 10.3);

2) у вікні Інспектора Об'єктів установити значення властивостей компоненти.

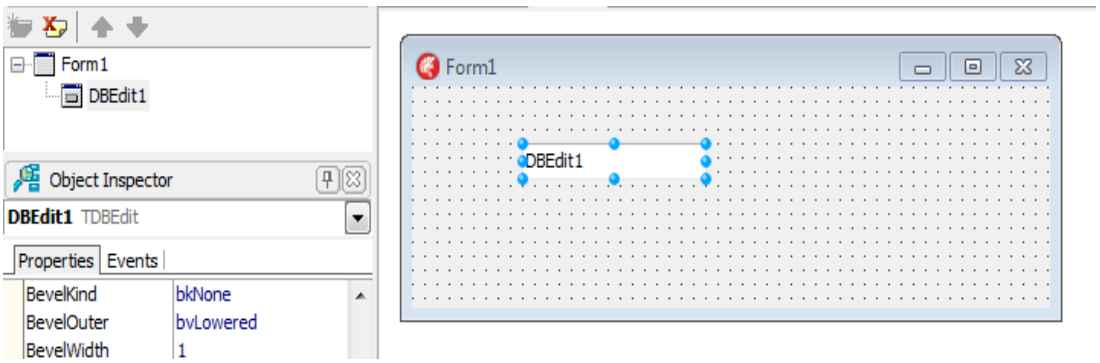


Рисунок 10.3 - До пояснення візуального проектування форми

На формі виділений компонент (після розміщення на формі він завжди виділений) обрамляється чорними прямокутними крапками. Шляхом протягання мишею за виділені крапки можна змінювати розміри компонента.

Вікно Інспектора Об'єктів (рис. 10.4) має дві сторінки: Properties (Властивості) і Events (Події).

На сторінці Properties відображаються основні властивості об'єкта (виділеного компонента). Більшість з них установлюються “за замовчуванням”.

На сторінці Events відображаються властивості-події, що зв'язані з даним компонентом. Для кожної властивості-події можна створити процедуру – оброблювач даної події. Ця процедура буде виконуватися щораз при настанні події.

Вікно Редактора коду після запуску середовища Delphi знаходиться за вікном Редактора форм і майже перекривається ім. Для переходу у вікно Редактора коду потрібно натиснути клавішу **F12**.

У вікні міститься вихідний текст модуля форми мовою Object Pascal. Спочатку в ньому міститься текст модуля Unit1.pas форми Form1. При візуальному проектуванні форми код генерується автоматично.

Клавіша **F12** дозволяє по черзі активізувати вікно Редактора форми чи вікно Редактора коду.

Клавіша **F11** дозволяє послідовно активізувати вікно Редактора Об'єктів, вікно Редактора форми чи вікно Редактора коду.

Контрольні запитання:

1. Принцип об’єктно-орієнтованого програмування.
2. Що таке об’єкт? Поняття класу.
3. Ієрархія основних класів *Delphi*.
4. Призначення базових класів.
5. Поняття компонента.
6. У чому полягає суть візуального програмування в *Delphi*?
7. Елементи інтерфейсу середовища *Delphi*: призначення та застосування.

11. Проект Delphi. Структура програми БД

11.1. Проект Delphi і його компіляція

11.1.1. Склад проекту

Додаток може містити довільне число форм і використовувати будь-як інтерфейс (MDI або SDI). Звичайно одна форма відповідає за виконання групи однорідних операцій, об'єднаних загальним призначенням. В основі будь-якого додатка баз даних лежать набори даних, що являють собою групи записів (їхній зручно представити у виді таблиць у пам'яті), переданих з бази даних у додаток для перегляду і редагування.

Створювана в середовищі *Delphi* програма складається з декількох елементів (файлів), що поєднуються в один **проект**. До складу проекту входять наступні файли:

файл проекту (.dpr) – текстовий файл, що містить інформацію про склад проекту й оператори ініціалізації і запуску програми на виконання;

файл форми (.dfm) – текстовий файл, що містить опис форми (склад форми, властивості форми і розміщених у ній компонент і ін.);

файл модуля (.pas) – текстовий файл, що містить код (текст програми мовою Object Pascal).

Файл модуля завжди створюється для кожної форми. Також може бути створений окремий файл модуля, не зв'язаний з формою.

файл описания проекта (.dproj) – текстовий файл, що містить поточні установки параметрів проекту (параметри компіляції, компонування);

файл конфігурації проекту (.cfg) – текстовий файл, призначений для збереження параметрів середовища Delphi;

файл ресурсів проекту (.res) – двійковій файл, що містить піктограму проекту й інші ресурси, використовувані в проекті (курсори й ін.)

файл параметрів проекту (.dof) – текстовий файл, що містить поточні установки параметрів проекту (параметри компіляції, компонування).

При запуску *Delphi* автоматично створюється проект з ім'ям за замовчуванням Project1. Цей проект має у своєму складі одну форму з ім'ям Form1.

При створенні проекту рекомендується на початку зберегти проект із потрібним ім'ям, а потім продовжувати його розробку. Усі файли проекту рекомендується зберігати в одній папці.

Для збереження проекту потрібно виконати команду меню **File**│**Save Project as**. При цьому послідовно будуть відкриватися два вікна:

1. У першому вікні (**Save Unit1 as**) потрібно установити папку, у якій будуть зберігатися усі файли проекту, і ввести *ім'я файлу модуля форми*. За замовчуванням пропонується ім'я Unit1.pas. Ви введіть - Lab6_Un1.pas (для лабораторної роботи 6) і натисніть кнопку **Зберегти**. Після цього відкриється друге вікно для збереження файлу проекту.

2. В другому вікні (**Save Project1 as**) потрібно ввести *ім'я файлу проекту*. За замовчуванням пропонується ім'я Project1.dpr. Ви введіть - Lab6.dpr і знов натисніть кнопку **Зберегти**.

Після виконання цих подій проект буде збережений.

У майбутньому, на практичних заняттях вам потрібно створити окремі папки: Lab6, Lab7,…, призначені для збереження проектів для кожної з лабораторних робот. Ці папки створюються у "вашій" папці, у якої ви зберігаєте учбову БД *Access*. Структура створюваних папок повинна відповідати рис. 11.1.

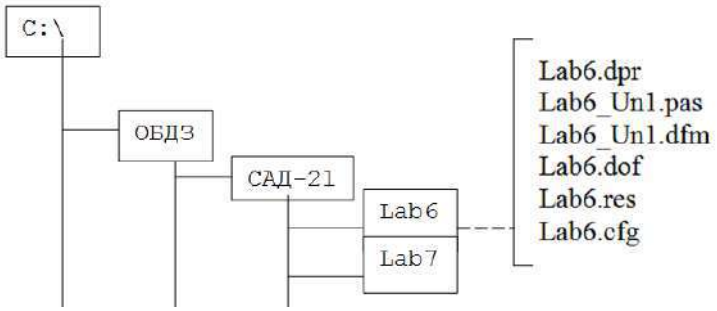


Рисунок. 11.1 - Структура папок для проектів

Типова помилка, що допускають студенти при виконанні лабораторних робіт – це збереження файлів проекту і модулів у різних папках. Щоб уникнути цієї помилки рекомендується закрити проект (не закриваючи середовище), перевірити наявність усіх файлів у папці, а потім знову відкрити проект через Головне меню: **File**│**Open Project...**

11.1.2. Компіляція і виконання проекту

Для того, щоб проект міг бути запущений на виконання, необхідно виконати його компіляцію. У результаті компіляції і наступної зборки створюється *exe*-файл, який виконується. Створений *exe*-файл, власне, і є отриманою в середовищі *Delphi* прикладною програмою.

Процес створення *exe*-файлу схематично зображений на Рис. 11.2.

Після першої компіляції і зборки в папці проекту повинні з'явитися ще такі файли:

- **відкомпільований файл модуля** (.dcu) – створюється для кожного вихідного файлу модуля (.pas);
- **файл, що виконується**, (.exe) - файл прикладної програми, що може виконуватися під керуванням ОС *Windows* без середовища *Delphi*.

Для тільки компіляції проекту необхідно виконати команду меню **Project**│**Compile Project** чи натиснути комбінацію клавіш «**Ctrl+F9**».

Для компіляції і виконання проекту потрібно виконати команду меню **Run**│**Run** чи натиснути клавішу «**F9».**

Для покрокового виконання програми, що буває необхідно під час її налагоджування, варто натиснути клавішу «**F8**».

Для примусового завершення програми потрібно натиснути комбінацію клавіш «**Ctrl+F2**».

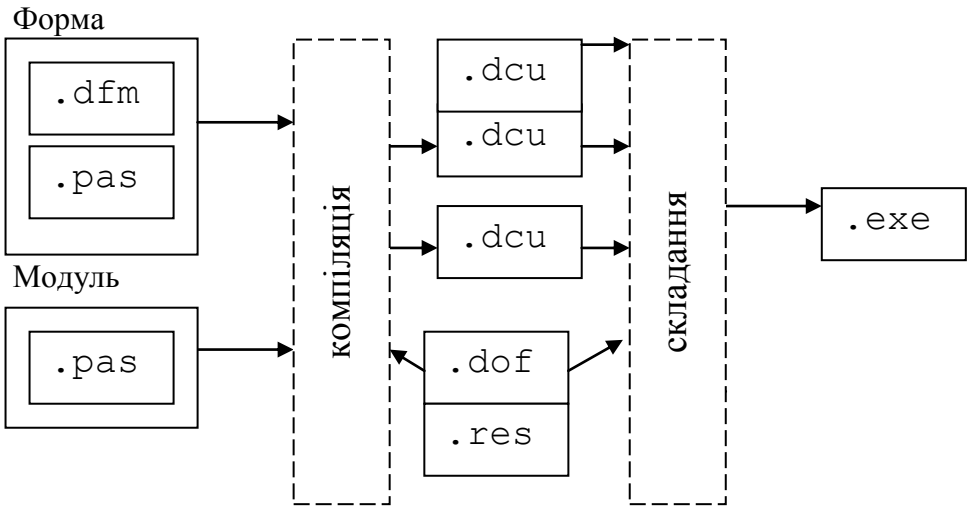


Рисунок. 11.2 - Схема процесу створення *exe*-файлу

11.1.3 Схема взаємодії програми Delphi із БД

Прикладна програма БД, створена в *Delphi* (і не тільки в *Delphi*), взаємодіє з БД через посередника – процесор БД **BDE** (Borland DataBase Endgine). **BDE** реалізує всі операції безпосереднього доступу до даних і керування даними. **BDE** являє собою набір DLL-бібліотек, що містять процедури і функції низькорівневого доступу до даних. На Рис. 11.3 зображена узагальнена схема взаємодії прикладної програми з локальними і віддаленими БД.

Коли програмі *Delphi* потрібно зв'язатися з БД, вона звертається до **BDE** і повідомляє йому **псевдонім** (зареєстроване ім'я) БД і ім'я потрібної таблиці. Псевдонім крім імені БД містить також інформацію про тип драйвера, що повинний використовуватися з цієї БД. **Драйвер** – це допоміжна програма, що реалізує операції доступу до даних конкретної СУБД, вона як би "знає", як звертатися до інформації БД цього типу. У **BDE** маються власні драйвери для найбільш розповсюджених СУБД: *Paradox, dBase, FoxPro, Access*.

Якщо в **BDE** є власний драйвер для даної СУБД, **BDE** відразу використовуючи його зв'язується з БД і потрібною таблицею в ній, обробляє запит користувача, і повертає в прикладну програму результати обробки. Якщо в **BDE** власного драйвера для потрібної СУБД немає, то можна використовувати драйвери **ODBC** (Open DataBase Connectivity – відкритий інтерфейс підключення до БД). **ODBC** – це також бібліотека DLL, по функціях подібна **BDE**, але розроблена *MicroSoft*. У **ODBC** *MicroSoft* включила драйвери для доступу практично до будь-яких відомих у даний час СУБД (*SyBase, MS SQL Server, Oracle, InterBase* і ін.). Завдяки цьому і програми *Delphi* можуть працювати з БД цих СУБД через драйвери **ODBC**.

Альтернативна **BDE**– це технологія **ADO** (Active Data Objects) яка надає інтерфейс користувача до будь-яких типів даних у реляційних і не реляційних БД, електронну пошту, системні, текстові і графічні дані. Зв'язок з даними здійснюється за допомогою технології OLE DB.

Альтернативні можливості доступу до БД розширені за рахунок технології *dbExpress*. Це набір драйверів, що забезпечують доступ до серверів *SQL* на основі єдиного інтерфейсу. Зараз використання **ADO є** основним і найбільш розповсюдженим способом доступу до БД.

Разом з **BDE** поставляється спеціальна програма-утиліта **BDE Administrator**, за допомогою якої можна переглядати, редагувати і створювати нові псевдоніми БД.

Звичайно програма **BDE Administrator** включається в папку "Панель керування".

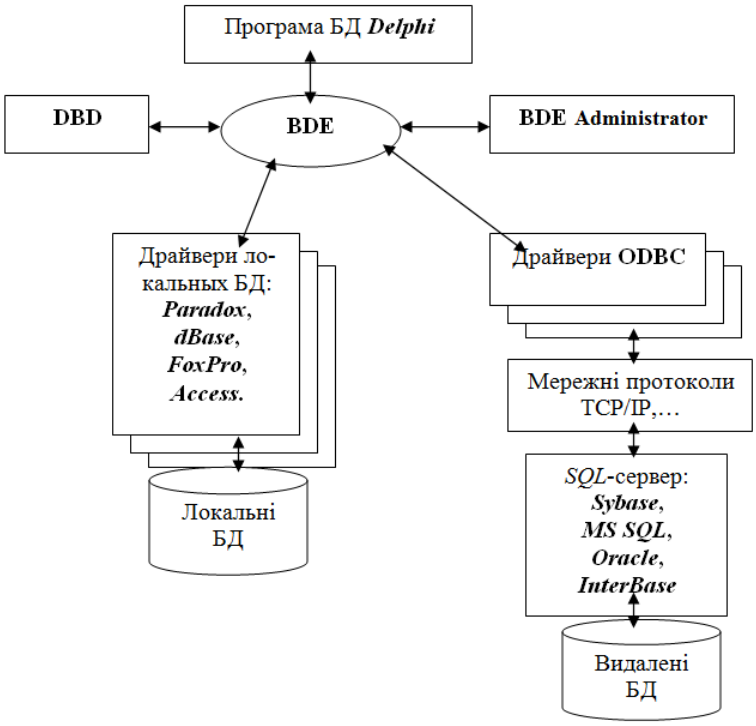


Рисунок 11.3 - Загальна схема взаємодії програми *Delphi* з БД

11.2 Основні компоненти Delphi для роботи з БД

Як ми вже знаємо, будь-яка прикладна програма *Delphi* будується на основі використання тих чи інших компонентів. Програма БД використовує спеціальні компоненти, призначені для роботи з БД. Серед них є як не візуальні, так і візуальні компоненти.

Не візуальні компоненти призначені для забезпечення доступу до даних і витягу з БД інформації, що потім може оброблятися прикладною програмою і відображатися у візуальних компонентах.

Основні не візуальні компоненти включені в загальну ієрархію класів *Delphi*. Фрагмент цієї ієрархії з найбільш важливими компонентами зображений на рис. 11.4.

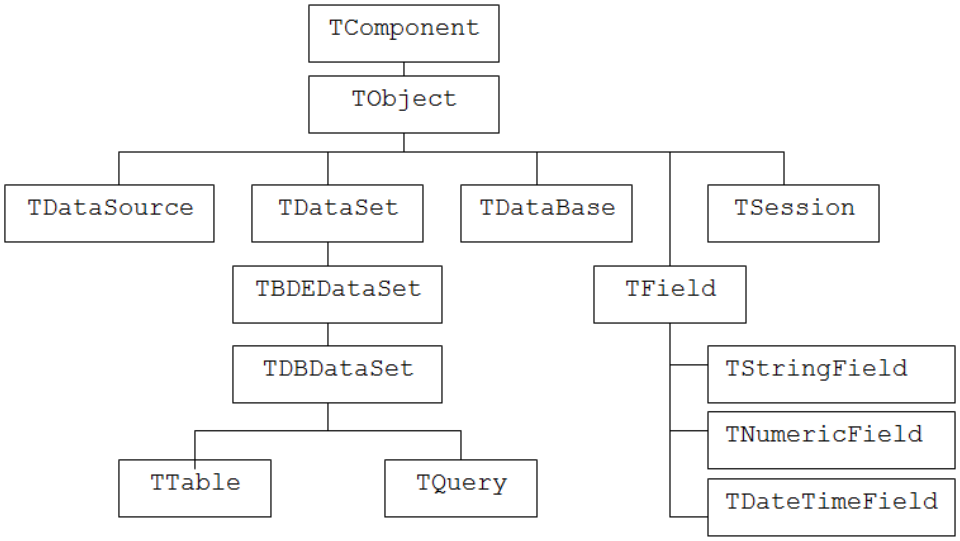


Рисунок 11.4 - Компоненти *Delphi* для роботи з БД

Розглянемо найбільш важливі класи компонентів, призначених для роботи з БД.

Модуль даних. Для розміщення компонентів доступу до даних у додатку баз даних бажано використовувати спеціальну "форму" — модуль даних (клас TDataModule). Модуль даних не має нічого загального зі звичайною формою додатка, адже його безпосереднім предком є клас TComponent. У модулі даних можна розміщати тільки невізуальні компоненти. Модуль даних доступний розроблювачеві, як і будь-який інший модуль проекту, на етапі розробки. Користувач додатка не може побачити модуль даних під час виконання.

Для створення модуля даних можна скористатися палітрою об'єктів або головним меню Delphi. Значок модуля даних Data Module розташований на сторінці New (рис.11.5).

Рисунок 11.5 - Розміщення модуля даних Data Module

Модуль даних має мало загального зі стандартною формою, хоча б тому, що клас TDataModule відбувається безпосередньо від класу TComponent. У нього майже цілком відсутні властивості і методи-оброблювачі подій, адже від платформи для інших невізуальних компонентів майже нічого не потрібно, хоча нащадки модуля даних, що працюють у розподілених додатках, виконують досить важливу роботу.

Для створення структури (моделі, діаграми) даних, з яким працює додаток, можна скористатися можливостями, наданими сторінкою Diagram Редактори коду.

Любою елемент з ієрархічного дерева компонентів модуля даних можна перенести на сторінку діаграми і задати зв'язку між ними. За допомогою керуючих кнопок можна задавати між елементами діаграми відносини синхронного перегляду і головний/підлеглий. При цьому виробляється автоматичне настроювання властивостей відповідних компонентів. Для звертання компонентів доступу до даних, розташованим у модулі даних, з інших модулів проекту необхідно включити ім'я модуля в секцію uses:

```
unit InterfaceModule;
...
 implementation
uses DataModule;
...
DataModule.Tablel.Open;
```

Перевагою розміщення компонентів доступу до даних у модулі даних є те, що зміна значення будь-якої властивості проявиться відразу ж у всіх звичайних модулях, до яких підключений цей модуль даних. Крім цього, всі оброблювачі подій цих компонентів, тобто вся логіка роботи з даними додатка, зібрані в одному місці, що теж досить зручно.

Кожен набір даних инкапсулирован у спеціальному компоненті доступу до даних. У VCL Delphi реалізований набір базових класів, що підтримують функціональність наборів даних, і практично ідентичні по складу набори дочірніх компонентів для технологій доступу до даних. Їхній загальний предок —клас TDataSet.

TDataSet – базовий клас, що забезпечує функціонування наборів даних у програмі *Delphi*. Під **набором даних (НД)** розуміється підмножина записів, отриманих з однієї чи декількох таблиць БД і розташована у оперативної пам'яти. Основне призначення класу **TDataSet** – визначити основні властивості і методи для роботи з реляційними наборами даних, що не залежать від **BDE**.

TBDEDataSet – безпосередній нащадок класу **TDataSet** , що забезпечує реалізацію найважливіших властивостей і методів роботи з НД за рахунок звертання до функцій **BDE.**

TDBDataSet – клас, у якому додається до успадкованих нові властивості і методи, що забезпечують з'єднання НД із БД. Цій клас є безпосереднім предком основних компонентів доступу до даних **TTable** і **TQuery**.

TTable – клас, що реалізує НД, джерелом якого є *одна таблиця* БД. Містить багато властивостей і методів, за допомогою яких можна виконувати над набором даних багатий спектр операцій. Більшість з них перевизначають властивості і методи, визначені в компонентах-предках **TDataSet** і **TBDEDataSet.**

TQuery – клас, цілком аналогічний класу **TTable**, однак відрізняється від нього тим, що може містити дані, отримані з *однієї чи декількох таблиць* БД. Зміст інформації у цьому НД визначається його властивістю, що є *SQL*-запитом.

TDataSource – клас компонента, що називається "джерелом даних" і призначений *для зв'язування* компонента НД (**TTable** чи **TQuery**) з візуальними компонентами.

TField – клас, що представляє окреме поле (стовпчик) НД. Містить властивості і методи, що визначають поведінку і параметри поля. Клас є абстрактним і безпосередньо ніколи не використовується. Він використовується як батьківський для створення похідних класів, що реалізують поля конкретних типів, наприклад:

TStringField – клас полів строкового типу;

TNumericField – клас полів числового типу;

TDateTimeField – клас полів типу “Дата/Час”, і ін.

Безпосередньо в прикладної програмі з розглянутих компонентів ми будемо використовувати тільки компоненти **TQuery**, **TTable** і **TDataSource**. Вони розміщаються на формі прикладної програми (чи в спеціальному модулі). Під час виконання програми вони не відображаються. Похідні від **TField** класи використовуються для доступу до полів із програмного коду.

Візуальні компоненти призначені для відображення, введення і редагування даних. Вони, по суті, забезпечують інтерфейс прикладної програми з користувачем. У більшості випадків вони являють собою модифікації стандартних елементів керування *Windows*, пристосованих для роботи з НД.

Найбільш розповсюдженими (ми їх будемо використовувати при виконанні лабораторних робіт) є наступні візуальні компоненти:

TDBText – відображає значення одного поля з поточного запису НД у режимі “тільки для читання” (змінюватися дані не можуть).

TDBEdit – забезпечує як перегляд, так і редагування значення поля у поточному запису даного НД.

TDBGrid – відображує на екрані зміст полів НД у виді таблиці, у якій рядки відповідають записам, а стовпці – полям НД. Це дуже могутній і зручний компонент, який ми будемо використовувати у всіх лабораторних роботах.

TDBNavigator – дозволяє здійснювати навігацію по НД, переводити НД у стани вставки, зміни, додавання запису, запам'ятовувати зміни у даних.

Отже, ми змогли тільки перерахувати і коротко охарактеризувати найбільш важливі компоненти, призначені для роботи з БД. Більш докладно ми ще познайомимося з ними в процесі наступних занять.

11.3 Структура програми БД Delphi

Будь-яка програма *Delphi*, яка призначена для роботи з БД, завжди містить у собі як мінімум три наступних компоненти:

- компонент, що є НД (**TTable** чи **TQuery**);
- компонент – джерело даних (**TDataSource**);
- який-небудь візуальний компонент для відображення і (чи) редагування даних (**TDBGrid**, **TDBEdit**, …).

З обліком цього раніше розглянуту схему зв'язку прикладної програми з БД тепер можна деталізувати в такий спосіб (рис. 11.6):

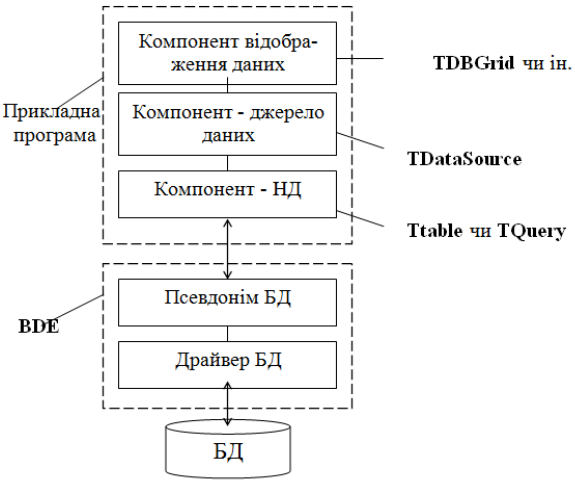


Рисунок 11.6 - Схема доступу до даних з прикладної програми БД *Delphi*

Компонент доступу до даних є основою додатка баз даних. На основі обраної таблиці БД він створює набір даних і дозволяє ефективно керувати їм. У процесі роботи такий компонент

тісно взаємодіє з функціями відповідної технології доступу до даних. Звичайно доступ до функціональності технології доступу до даних здійснюється через сукупність інтерфейсів. Усі компоненти доступу до даних є невізуальними.

Для створення нового проекту досить вибрати команду ***New \Application*** з меню ***File*** або скористатися Репозиторием об'єктів, що відкривається командою New з меню File.

У реальних проектах для розміщення компонентів доступу до даних варто використовувати модуль даних. Потім на форму нового проекту необхідно перенести компонент, инкапсулирующий набір даних, і виконати наступні дії.

Відкриття і закриття набору даних можна передбачити як реакцію на дії користувача або виникнення події. Найчастіше набір даних повинний відкриватися при першому показі форми і закриватися при її закритті.

Компонент **TTable** створює в оперативній пам'яті "образ" реального НД (таблиці), з яким і виконуються всі подальші дії над даними в програмі.

Якщо використовується компонент **TQuery**, то НД може містити дані з декількох таблиць відповідно до заданого у компоненті *SQL*-запиту.

Для забезпечення зв'язку набору даних з візуальними компонентами відображення даних використовується спеціальний компонент TDataSource. Його роль полягає в керуванні потоками даних між набором даних і зв'язаними з ним компонентами відображення даних. Цей компонент забезпечує передачу даних у візуальні компоненти і повернення результатів редагування в набір даних, відповідає за зміну стану візуальних компонентів при зміні стану набору даних, передає сигнали керування від користувача (візуальних компонентів) у набір даних. Компонент TDataSource розташований на сторінці Data Access Палітри компонентів.

Компонент **TDataSource** керує потоком даних між НД і зв'язаним з ним компонентом відображення даних. Він синхронізує стан компонента відображення даних з поточним станом НД. Наприклад, при переміщенні по записах НД поточні значення полів у компоненті відображення даних автоматично обновляються.

Ще одна функція компонента TDataSource полягає в синхронізації поводження компонентів відображення даних зі станом набору даних. Наприклад, якщо набір даних не активний, то компонент TDataSource забезпечує видалення даних з компонентів відображення даних і їхній переклад у неактивний стан. Або, якщо набір даних працює в режимі "тільки для читання", те компонент TDataSource зобов'язаний передати в компоненти відображення даних заборона на зміну даних.

З одним компонентом TDataSource можуть бути зв'язані кілька візуальних компонентів відображення даних. Ці компоненти являють собою модифіковані елементи керування, що призначені для показу інформації з наборів даних.

При відкритті набору даних компонентів забезпечує передачу в набір даних записів з необхідної таблиці БД. Курсор набору даних установлюється на перший запис. Компонент TDataSource організує передачу в компоненти відображення даних значень необхідних полів з поточного запису. При переміщенні по записах набору даних поточні значення полів у компонентах відображення даних автоматично обновляються.

Користувач за допомогою компонентів відображення даних може переглядати і редагувати дані. Змінені значення відразу ж передаються з елемента керування в набір даних за допомогою компонента TDataSource. Потім зміни можуть бути передані в базу даних або скасовані.

Як тільки компонент - НД (**TTable** чи **TQuery**) одержує команду зв'язатися з БД, то далі відбувається наступне:

- Компонент - НД пересилає відповідний запит до **BDE**;
- **BDE** за інформацією псевдоніма БД визначає тип необхідного драйвера, виділяє для нового процесу необхідні ресурси, робить трансляцію запиту відповідно до особливостей драйвера використовуваної СУБД;
- При роботі з локальною БД драйвер безпосередньо звертається до таблиць БД (якщо виконується звертання до SQL-сервера, то викликається ще відповідне клієнтське програмне забезпечення сервера);
- Результат виконання запиту повертається в компонент - НД (**TTable** чи **TQuery**).

11.4. Етапи створення прикладної програми БД

Тепер розглянемо найбільш важливі моменти технології проектування прикладної програми БД. Вище ми вже встановили, що будь-яка програма БД повинна містити, як мінімум, три компоненти:

- компонент - НД, у якому містяться дані з якої-небудь таблиці;
- компонент - джерело даних, призначений для передачі даних між компонентом - НД і компонентами відображення даних;
- компонент відображення даних (візуальний компонент).

Таких “трійок” компонентів у прикладної програми може бути багато, однак, спосіб їхнього включення в програму і настроювання однаковий. Відповідно до цього розглянемо три етапи, що полягають у розміщенні і настроюванні цих трьох типів компонентів.

Для зручності і наочності викладу будемо вирішувати конкретну задачу – будемо розглядати приклад розміщення компонентів для відображення списку студентів з таблиці СТУДЕНТИ навчальної БД.

Етап 1. Розміщення і настроювання компонента TTable

1) Помістити на форму компонент **TTable** (зі сторінки Data Access палітри компонентів);

2) У вікні Інспектора Об'єктів для властивості DataBaseName установити (вибрати зі списку, що випадає) значення псевдоніма навчальної БД "UchProcess_AD21", що вже зареєстроване в BDE;

3) У властивості TableName установити ім'я таблиці, з якою повинний бути зв'язаний даний компонент. Зі списку, що випадає, виберіть ім'я таблиці СТУДЕНТИ;

4) У властивості Name увести з клавіатури ім'я компонента. За замовчуванням пропонується ім'я Table1. Для таблиці СТУДЕНТИ введемо Table_Stud;

5) У властивості Active установити значення True (вибором зі списку, що випадає, чи подвійним клацанням миші). Якщо це вдалося зробити, це значить, що зв'язок із БД встановлений, тобто у компонент поміщені дані з таблиці СТУДЕНТИ.

На рис. 11.7 показаний вид екрана з вікном Інспектора Об’єктів і вікном Редактора форми при розміщенні компонента Table_Stud.

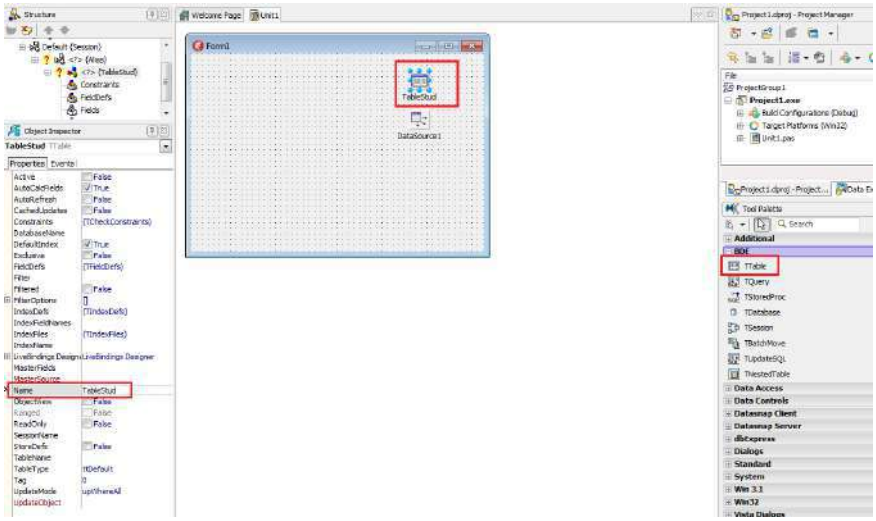


Рисунок 11.7 - Розміщення компонента Table1

Етап 2. Розміщення і настроювання компонента TDataSource

1) Помістити на формі компонент **TDataSource** (сторінка Data Access);

2) У властивості ***DataSet*** установити (шляхом вибору зі списку, що випадає) ім'я компонента - НД, з яким повинне бути зв'язане дане джерело. Установите значення Table_Stud;

3) Перейменувати компонент. Це не обов'язкова дія. Проте бажано привласнювати компонентам осмислені імена, що відповідають назвам зв'язаних наборів даних. Звичайна назва компонента комбінує ім'я набору даних .

Компонент TDataSource можна підключити не тільки до набору даних з тієї ж

форми, але і будь-який іншої, модуль якої зазначений у секції uses.

У властивості ***Name*** увести з клавіатури ім'я компонента. За замовчуванням

пропонується DataSource1. Уведіть значення DataSource_Stud.

Найчастіше одному наборові даних відповідає один компонент TDataSource, хоча них може бути кілька.

Компонент TDataSource має ряд корисних властивостей і методів.

Отже, зв'язування з компонентом набору даних виконує властивість property DataSet: TDataSet, а визначити поточний стан набору даних можна,

використовувавши властивість

type TDataSetState = (

dslnactive, dsBrowse, dsEdit, dslnsert, dsSetKey, dsCalcFields, dsFilter, dsNewValue, dsOldValue, dsCurValue, dsBlockRead, dsInternalCalc);

property State: TDataSetState.

За допомогою властивості property Enabled: Boolean можна включити або відключити всі зв'язані візуальні компоненти. При значенні False жоден зв'язаний компонент відображення даних не буде працювати.

Властивість property AutoEdit: Boolean, при значенні True завжди буде переводити набір даних у режим редагування при одержанні фокуса одним зі зв'язаних візуальних компонентів.

Аналогічно, метод procedure Edit; переводить зв'язаний набір даних у режим редагування.

Метод function IsLinkedTo(DataSet: TDataSet): Boolean; повертає значення True, якщо компонент, зазначений у параметрі DataSet, дійсно зв'язаний з даним компонентом TDataSource.

Метод-оброблювач type TDataChangeEvent = procedure(Sender: TObject; Field: TField) of object; property OnDataChange: TDataChangeEvent; викликається при редагуванні даних в одному зі зв'язаних візуальних компонентів.

Метод-оброблювач property OnUpdateData: TNotifyEvent; викликається перед збереженням змін у базі даних.

Метод-оброблювач property OnStateChange: TNotifyEvent; викликається при зміні стану зв'язаного набору даних

Етап 3. Розміщення і настроювання компонента відображення даних (на прикладі TDBGrid)

Для кожного візуального компонента відображення даних необхідно виконати наступні операції:

1. Помістити на формі компонент TDBGrid (сторінка Data Controls).

Зв'язати компонент відображення даних і компонент TDataSource. Для цього використовується властивість Datasource, що повинна вказувати на екземпляр необхідного компонента TDataSource. Один компонент відображення даних можна зв'язати тільки з одним компонентом TDataSource. Необхідний компонент можна вибрати в списку властивостей в Інспекторі об'єктів. У властивості DataSource вибрати зі списку, що випадає, ім'я джерела даних, з яким потрібно зв'язати компонент DBGrid_Stud. У списку, що випадає, виберіть компонент – джерело DataSource_Stud.

У властивості Name увести з клавіатури ім'я компонента. За замовчуванням пропонується DBGrid1. Уведіть значення DBGrid_Stud.

2. Задати поле даних. Для цього використовується властивість DataField типу TFields. У ньому необхідно вказати ім'я полю чи зв'язаного набору даних. Після завдання властивості Datasource поле можна вибрати зі списку. Цей етап застосовується тільки для компонентів, що відображають єдине поле.

Після цього в сітці DBGrid_Stud повинні з'явитися реальні дані з таблиці СТУДЕНТИ (але тільки за умови, якщо в компоненті Table_Stud раніше властивість ***Active*** була встановлена в значення *True*). Після виконання цього етапу вид екрана комп'ютера може бути такий, як це показано на рис. 11.8.

Відкриття і закриття набору даних можна передбачити як реакцію на дії користувача або виникнення події. Найчастіше набір даних повинний відкриватися при першому показі форми і закриватися при її закритті.

Листинг 1. Секція Implementation головного модуля проекту

```
implementation
{$R *.DFM}
procedure TForml.FormShowfSender: TObject);
begin
 try
Table_stud.Open;
 except
ShowMessage('Table open error');
 end;
end;
procedure TForml.FormClose(Sender: TObject; var Action: TCloseAction);
begin
Table_stud.Close;
end;
end.
```

При відкритті форми виконується метод оброблювач **FormShow**. У ньому набір даних відкривається за допомогою методу **Open**. Зверніть увагу на використання конструкції **try..except**, що забезпечує коректне завершення при виникненні виняткових ситуацій. Тому що помилки в роботі додатків баз даних можуть привести до серйозних наслідків (втрата або перекручування даних), те захисний код повинний бути присутнім у всіх критичних місцях. У методі-оброблювачі **FormClose**, що викликається при закритті форми, набір даних закривається методом **close**. Для виконання розглянутих операцій можна скористатися і властивістю Active. Однак реальні операції виконують зазначені методи. Тому використання властивості є зайвим етапом, та й з погляду ООП усі дії повинні виконувати методи об'єкта, а властивості служать тільки для представлення значень. Властивість Active сигналізує стан набору даних.

Тепер можна відкомпілювати проект і переконатися, що в сітці DBGrid_Stud можна переглядати і редагувати дані з таблиці СТУДЕНТИ створеної раніше навчальної БД. На рис. 11.8 зображено схему зв'язків між компонентами, яка створюється в результаті дій, виконаних у розглянутих вище етапах.

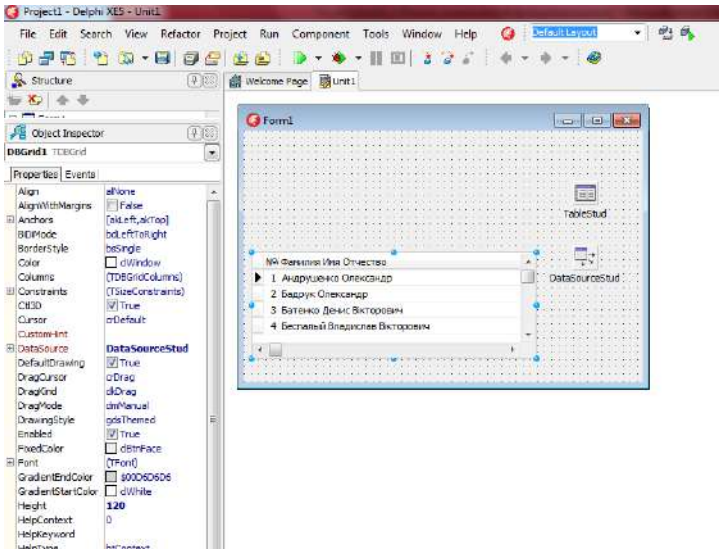


Рисунок 11.8 - Розміщення компонента DBGrid_Stud

Зв’язок візуального компонента з компонентом – джерелом даних (**TDataSourse**) встановлюється завдяки того, що у властивість ***DataSource*** записується ім’я (***Name***) компонента – дже-

рела даних. Зв’язок компонента **TDataSourse** з компонентом – НД (**TTable** чи **TQuery**) встановлюється через властивість ***DataSet*** , в якої записується ім’я (***Name***) компонента – НД. Зв’язок компонента – НД з БД встановлюється через властивість ***DataBaseName*** , що є псевдонімом БД. На Рис. 11.9 зображено компонент **TDataBase** (за замовчуванням його ім'я **DataBase1**), який ми раніше не згадували. Цей компонент завжди автоматично включається в будь-яку програму *Delphi*, яка працює з БД. Для розглянутої нами найпростішої програми установлювати якінебудь його властивості цього компонента **TDataBase** не потрібно.

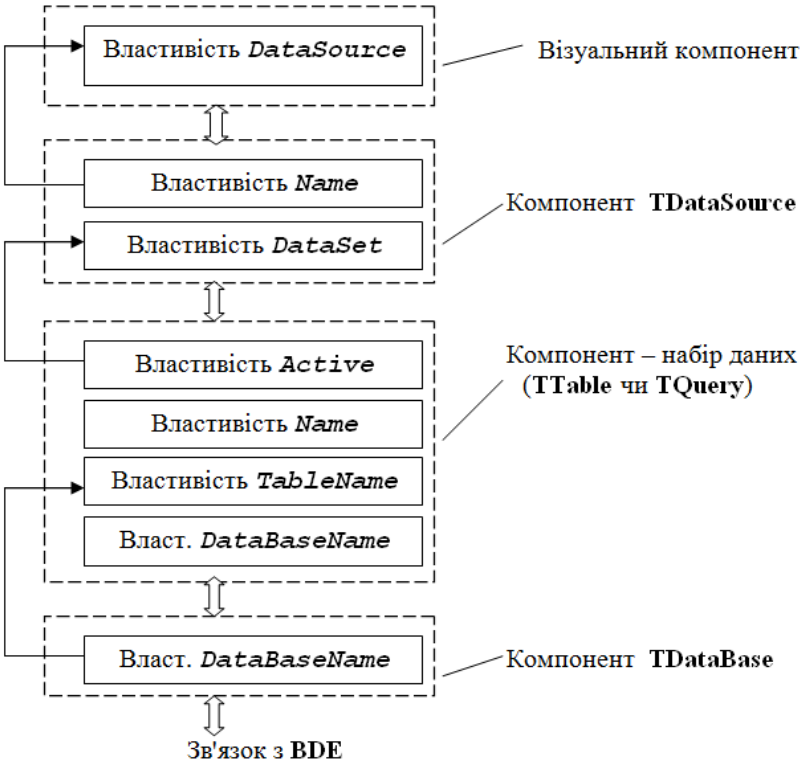


Рисунок 11.8 - Схема зв'язку між компонентами програми БД

11.5. Використання технології ADO до доступу к БД

Поряд із традиційними інструментами доступу до даних Borland Database Engine і ODBC у додатках Delphi можна застосовувати технологію **Microsoft Active Data Objects (ADO),** що заснована на можливостях COM, а саме інтерфейсів OLE DB.

Технологія ADO завоювала популярність у розроблювачів, завдяки універсальності — базовий набір інтерфейсів OLE DB мається в кожній сучасній операційній системі Microsoft. Тому для забезпечення доступу додатка до даних досить лише правильно вказати провайдер з'єднання ADO і потім переносити програму на будь-який

комп'ютер, де мається необхідна база даних і, звичайно, установлена ADO.

У Палітрі компонентів Delphi є сторінка ADO, що містить набір компонентів, що дозволяють створювати повноцінні додатки БД, що звертаються до даних через ADO.

Технологія Microsoft Active Data Objects забезпечує універсальний доступ до джерел даних з додатків БД. Таку можливість надають функції набору інтерфейсів, створені на основі загальної моделі об'єктів COM і описані в специфікації OLE DB.

Технологія ADO і інтерфейси OLE DB забезпечують для додатків єдиний спосіб доступу до джерел даних різних типів. Наприклад, додаток, що використовує ADO, може застосовувати однаково складні операції і до даних, що зберігається на корпоративному сервері SQL, і до електронних таблиць, і локальним СУБД. Запит SQL, спрямований будь-якому джерелу даних через ADO, буде виконаний.

Виникає питання: яким образом джерела даних зможуть виконати цей запит?

OLE DB являє собою набір спеціалізованих об'єктів COM, инкапсулирующих стандартні функції обробки даних, і спеціалізовані функції конкретних джерел даних і інтерфейсів, що забезпечують передачу даних між об'єктами.

Відповідно до термінології ADO, будь-яке джерело даних (база даних, електронна таблиця, файл) називається сховищем даних, з яким за допомогою провайдеру даних взаємодіє додаток.

Мінімальний набір компонентів додатка може включати об'єкт з'єднання, об'єкт набору даних, об'єкт процесора запитів.

Об'єкти OLE DB створюються і функціонують так само, як і інші об'єкти COM. Кожному об'єктові відповідає ідентифікатор класу CLSID, що зберігається в системному реєстрі. Об'єктові відповідає набір інтерфейсів, до методів яких можна звертатися після створення об'єкта.

У результаті додаток звертається не прямо до джерела даних, а до об'єкта OLE DB, що "уміє" представити дані (наприклад, з файлу електронної пошти) у виді таблиці БД або результату виконання запиту SQL.

Технологія ADO у цілому містить у собі не тільки самі об'єкти OLE DB, але і механізми, що забезпечують взаємодію об'єктів з даними і додатками. На цьому рівні найважливішу роль грають провайдери ADO, що координують роботу додатків зі сховищами даних різних типів.

Така архітектура дозволяє зробити набір об'єктів і інтерфейсів відкритим і розширюваним. Набір об'єктів і відповідний провайдер може бути створений для будь-якого сховища даних без внесення змін у вихідну структуру ADO. При цьому істотно розширюється саме поняття даних — адже можна розробити набір об'єктів і інтерфейсів і для нетрадиційних табличних даних. Наприклад, це можуть бути графічні дані геоинформационных систем, деревоподібні структури із системних реєстрів, дані CASE-інструментів і т.д.

Тому що технологія ADO заснована на стандартних інтерфейсах COM, що є системним механізмом Windows, це скорочує загальний обсяг працюючого програмного коду і дозволяє поширювати додатка БД без допоміжних програм і бібліотек.

Нижченаведений опис специфікації OLE DB представлено відповідно до офіційної термінології Microsoft для даної предметної області.

Специфікація OLE DB розрізняє наступні типи об'єктів, що будуть розглянуті нижче.

Перечисельник (Enumerator) виконує пошук джерел даних або інших перечисельників. Використовується для забезпечення функціонування провайдерів ADO.

Об'єкт-джерело даних (Data Source Object) представляє сховище даних.

Сесія (Session) поєднує сукупність об'єктів, що звертаються до одного сховища даних.

Транзакция (Trasaction) инкапсулирует механізм виконання транзакции.

Команда (Command) містить текст команди і забезпечує її виконання. Командою може бути запит SQL, звертання до таблиці БД і т.д.

Набір рядів (Rowset) являє собою сукупність рядків даних, що є результатом виконання команди ADO.

Об'єкт-помилка (Error) містить інформацію про виняткову ситуацію.

Розглянемо функціональні можливості основних об'єктів і інтерфейсів OLE DB.

Об'єкти- перечисельники забезпечують пошук будь-яких об'єктів ADO, що мають доступ до джерел даних. При цьому інші перечисельники також видні в даному перечисельнику. Первинний пошук джерел даних здійснюється в провайдеру ADO. Перечисельники можуть відбирати тільки джерела даних конкретних типів, тому провайдер забезпечує доступ до конкретному типові сховища даних.

У складі ADO мається системний кореневий перечисельник, що виконує початковий пошук інших перечисельників і джерел даних. Його можна використовувати, знаючи його ідентифікатор класу CLSID_OLEDB_ENUMERATOR.

Функції перечисельника утримуються в інтерфейсі isourcesRowset. Метод function GetSourcesRowset(const punkOuter: lUnknown; const riid: TGUID; cPropertySets: UINT; rgProperties: PDBPropSetArray; out ppSourcesRowset: lUnknown): HResult; stdcall; повертає посилання на об'єкт набору рядів (див. вище), що містить зведення про знайдені джерела даних або перечисельниках.

Внутрішній механізм ADO, що забезпечує з'єднання зі сховищем даних, використовує два типи об'єктів. Це об'єкти-джерела даних і об'єкти-сесії.

Об'єкт-джерело даних забезпечує представлення інформації про необхідне реальне джерело даних і підключення до нього. Для введення зведень про сховище даних використовується інтерфейс iDBProperties. Для успішного підключення необхідно задати обов'язкові зведення. Імовірно, для будь-якого сховища даних буде актуальної інформація про його ім'я, користувача і пароль. Однак кожен тип сховища має власні унікальні настроювання. Для одержання списку всіх обов'язкових параметрів з'єднання з даним сховищем можна скористатися методом functio який повертає заповнену структуру DBPROPINFO.

Для кожного обов'язкового параметра в елементі dwFlags установлюється значення DBPROPFLAGS_REQUIRED.

Для ініціалізації з'єднання необхідно використовувати метод function Initialize інтерфейсу iDBinitiaiize об'єкта-джерела даних.

З об'єкта-джерела даних можна створювати об'єктів-сесії. Для цього використовується метод function CreateSession. Сесія призначена для забезпечення роботи транзакций і наборів рядів.

Керування транзакциями в OLE DB реалізовано на двох рівнях. По-перше, усіма необхідними методами володіє об'єкт сесії. Він має інтерфейси ITransaction, ITransactionJoin, ITransactionLocal, ITransactionObject. Усередині сесії транзакция керується інтерфейсами ITransactionLocal, ItransactionSC, ITransaction і їхніми методами StartTransaction, Commit, Rollback. По-друге, для об'єкта сесії можна створити об'єкт транзакции за допомогою методу function GetTransactionObject інтерфейсу ITransactionObject, що повертає посилання на інтерфейс объекта-транзакции.

Об'єкт-набір рядів є основним об'єктом ADO, що забезпечує роботу з даними. Він инкапсулирует сукупність рядів із джерела даних, механізми навігації по рядах і

підтримки рядів в актуальному стані.

Об'єкт сесії має обов'язковий інтерфейс IOpenRowset з методом function OpenRowset який відкриває необхідний набір рядів. У залежності від можливостей джерела даних набір рядів може підтримувати різні інтерфейси. Але п'ять з них є обов'язковими:

IRowset — забезпечує навігацію по рядах;

IAccessor — забезпечує представлення інформації про формат рядів, що утримуються в буфері набору рядів;

IRowsetinfo — дозволяє одержати інформацію про набори рядів (наприклад, число рядів або число обновлених рядів);

Icoiumnsinfo — дозволяє одержати інформацію про колонків рядів (найменування, тип даних, можливість відновлення і т.д.);

IconvertType — містить єдиний метод canConvert, що дозволяє визначити можливість перетворення типів даних у наборі рядів.

На відміну від звичної практики розробки інтерфейсів у рамках моделі COM, інтерфейси OLE DB часто мають один-два методу. У результаті велика група інтерфейсів реалізує трохи цілком стандартних функцій.

Додаткові можливості по керуванню набором рядів надають наступні інтерфейси:

IRowsetchange — виконує зміни в наборі рядів (вносить зміни, додає нові ряди, видаляє ряди і т.д.);

IRowsetidentity — дозволяє порівнювати ряди різних рядів;

IRowsetindex — забезпечує використання індексів;

IRowsetLocate — виконує пошук у наборі рядів;

IRowsetupdate — реалізує механізм кэширования змін.

Програмні засоби ADO були б неповними, якби не мали можливості використовувати для роботи з даними мова SQL. Оператори DML і DDL, ряд спеціальних операторів ADO носять загальні назви текстових команд.

Об'єкт-команда інкапсулируе саму текстову команду і механізм обробки і передачі команди. Об'єкт команди виконує наступні операції:

- розбір тексту команди;

- зв'язування команди з джерелом даних;
- оптимізацію команди;
- передачу команди джерелу даних.

Крім основного, об'єкт команди забезпечує доступ до додаткових інтерфейсів:

ICommandPrepare — містить два методи (Prepare И Unprepare) для підготовки команди;

icommandProperties — задає для команди властивості, що повинні підтримуватися набором даних, що повертається командою;

iCommandText — керує текстом команди (цей інтерфейс обов'язковий для об'єкта команди);

icommandwithParameters — забезпечує роботу з параметрами команди.

Провайдери ADO забезпечують з'єднання додатка, що використовує дані через ADO, із джерелом даних (сервером SQL, локальної СУБД, файловою системою і т.д.). Для кожного типу сховища даних повинний існувати провайдер ADO.

Провайдер "знає" про місце розташування сховища даних і його змісті, уміє звертатися до даних із запитами й інтерпретувати службову інформацію, що повертається, і результати запитів з метою їхньої передачі додаткові. Список встановлених у даній операційній системі провайдерів доступний для вибору при установці з'єднання через компонент TADOConnection.

При інсталяції Microsoft Active Data Objects в операційній системі встановлюються наступні стандартні провайдери.

Microsoft Jet OLE DB Provider забезпечує з'єднання з даними СУБД Access за допомогою технології BAT.

Microsoft OLE DB Provider for Microsoft Indexing Service забезпечує доступ тільки для читання до файлів і Internet-ресурсів Microsoft Indexing Service.

Microsoft OLE DB Provider for Microsoft Active Directory Service забезпечує доступ до ресурсів служби каталогів (Active Directory Service).

Microsoft OLE DB Provider for Internet Publishing дозволяє використовувати ресурси, надані Microsoft FrontPage,

Microsoft Internet Information Server, HTTP-файли.

Microsoft Data Shaping Service for OLE DB дозволяє використовувати ієрархічні набори даних.

Microsoft OLE DB Simple Provider призначений для організації доступу до джерел даних, підтримуючі тільки базисні можливості OLE DB.

Microsoft OLE DB Provider for ODBC drivers забезпечує доступ до даних, що уже "прописані" за допомогою драйверів ODBC.

Microsoft OLE DB Provider for Oracle забезпечує з'єднання із сервером Oracle.

Microsoft OLE DB Provider for SQL Server забезпечує з'єднання із сервером Microsoft SQL Server.

Механізм доступу до даних через ADO і численні об'єкти й інтерфейси реалізовані в VCL Delphi у виді набору компонентів, розташованих на сторінці ADO.

Компонент TADOConnection увібрав можливості перечисельника, джерела даних і сесії з можливостями обслуговування транзакций.

Текстові команди ADO реалізовані в компоненті TADOCommand.

Набори рядів можна одержати за допомогою компонентів TADOTable, TADOQuery, TADOStoredProc. Кожний з них реалізує спосіб доступу до конкретного типу представлення даних у сховище.

Набір властивостей і методів компонентів ADO забезпечує реалізацію всіх необхідних додаткові БД функцій. Способи використання компонентів ADO деяким відрізняються від стандартних компонентів VCL доступу до даних. Однак при необхідності розроблювач може використовувати всі можливості інтерфейсів ADO, звертаючи до них через відповідні об'єкти ADO. Посилання на об'єкти маються в компонентах.

Компоненти доступу до даних ADO можуть використовувати два варіанти підключення до сховища даних. Це стандартний метод ADO і стандартний метод Delphi.

У першому випадку компонента використовують властивість connectionstring для прямого звертання до сховища даних. В другому випадку використовується спеціальний компонент

TADOConnection, що забезпечує розширене керування з'єднанням і дозволяє звертатися до даного декількох компонентам одночасно.

Властивість connectionstring призначена для збереження інформації про з'єднання з об'єктом ADO. У ньому через крапку з коми перелічуються всі необхідні параметри. Як мінімум, це повинніо бути імена провайдеру з'єднання або вилученого сервера. При необхідності вказуються шлях до вилученого провайдеру і параметри, необхідні провайдерові: 'User Name=User_Name;Password=Password';

Кожен компонент, що звертається до сховища даних ADO самостійно, задаючи параметри з'єднання у властивості Connectionstring, відкриває власне з'єднання. Чим більше додаток містить компонентів ADO, тим більше з'єднань може бути відкрито одночасно.

Компонент TADOConnection призначений для керування з'єднанням з об'єктами сховища даних ADO. Він забезпечує доступ до сховища даних компонентам ADO, инкапсулирующим набір даних. Застосування цього компонента дає розроблювачеві ряд переваг: усі компоненти доступу до даних ADO звертаються до сховища даних через одне з'єднання; можливість прямо задати об'єкт провайдеру з'єднання; доступ до об'єкта з'єднання ADO; можливість виконувати команди ADO; виконання транзакций; розширене керування з'єднанням за допомогою методів-оброблювачів подій.

Перед відкриттям з'єднання необхідно задати його параметри. Для цього призначена властивість property ConnectionString: WideString (рис.11.9).

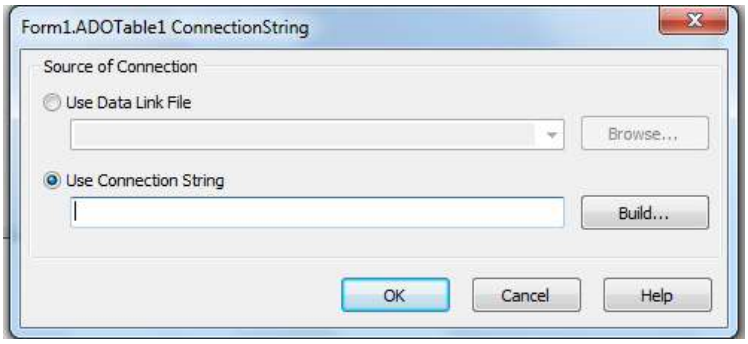


Рисунок 11.9 - Редактор настроювання з'єднання ADO

Набір параметрів змінюється в залежності від типу провайдеру і може набудовуватися як вручну, так і за допомогою спеціального редактора параметрів з'єднання, що викликається подвійним щигликом на компоненті TADOConnection, перенесеним на форму, або щигликом на кнопці в поле редагування властивості ConnectionString в Інспекторі об'єктів (рис.11.10).

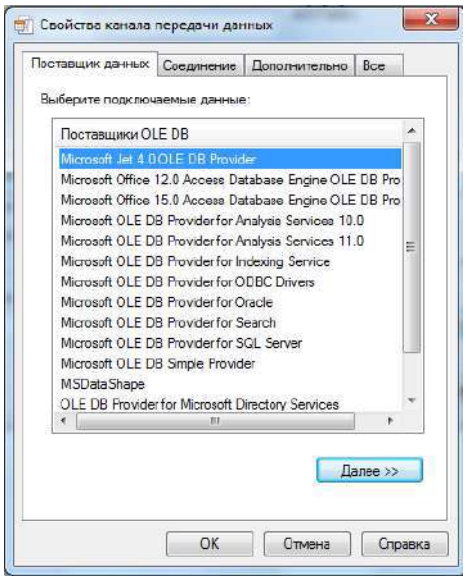


Рисунок 11.10 - Вікно вибору провайдера OLE DB

На сторінки Provider потрібно вибрати провайдер **Microsoft Jet 4.0 OLE DB Provider** для доступу к БД Access, На вкладці Connection ввести путь до БД (рис.11.11) та натиснути кнопку Test Connection. Якщо путь вибран правильно то з'єднання встановлено.

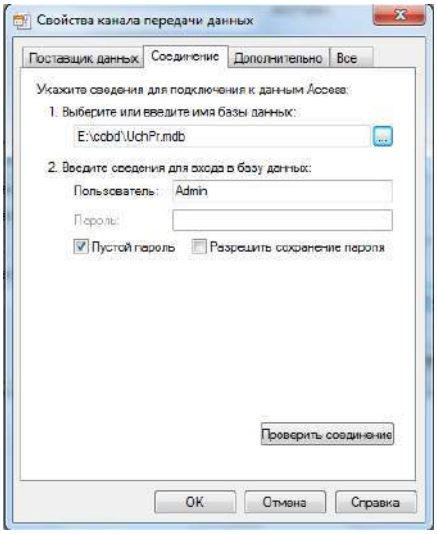


Рис.11.11. Вікно Data Link Properties

Контрольні запитання:

1. Основні компоненти, яки обов’язково містить програма *Delphi*, що призначена для роботи з БД.
2. Як встановлюється зв’язок програми з БД?
3. Призначення і установка основних властивостей компонента **TTable**.
4. Призначення і установка основних властивостей компонента **TDataSource**.
5. Призначення і установка основних властивостей візуальних компонентів.
6. Технологія ADO: призначення, гідності, недоліки.
7. Компоненти - набори даних ADO: TADODataSet, TADOTable, TADOQuery.

12. Компоненти – набори даних

12.1. Поняття НД. Основні властивості, методи, події

12.1.1. Компоненти – набори даних

Під **набором даних** (НД) розуміється група записів з даними, узятими з однієї чи декількох таблиць. У програмах *Delphi* НД створюються за допомогою компонентів **TTable** і **TQuery**. Тому компоненти **TTable** і **TQuery** часто називають просто наборами даних.

Компонент **TTable** завжди представляє дані, узяті з однієї таблиці. Компонент **TTable** використовується звичайно при роботі з локальними БД.

Компонент **TQuery** являє собою НД, отриманий на основі *SQL*-запиту. Тому в ньому можуть представлятися дані, узяті з декількох таблиць. Компонент **TQuery** рекомендується використовувати при роботі з віддаленими БД в архітектурі "клієнт-сервер".

Компоненти **TTable** і **TQuery** є спадкоємцями класу **TDataSet**, у якому визначені всі найважливіші властивості, методи і події, необхідні для роботи з НД. Нижче розглядаються найбільш важливі з них. Усі вони рівною мірою доступні в компонентах **TTable** і **TQuery.**

12.1.2. Огляд найбільш важливих властивостей, методів і подій

Властивість ***State*** визначає поточний стан НД. Воно може приймати наступні значення:

dsInactivate – НД закритий, дані недоступні;

dsBrowse – дані можна переглядати, але не можна змінювати;

dsEdit – дані поточної запису можна редагувати;

dsInsert – може вставлятися новий запис;

dsSetKey – НД знаходиться в режимі пошуку записів;

dsCalcFields – виконується установка значень полів, що обчислюються, (по алгоритму, що заданий в оброблювачі події ***OnCalcFields***);

dsFilter – НД знаходиться в режимі фільтрації записів (для поточного запису перевіряється умова фільтрації, що задана в оброблювачі події ***OnFilterRecord***).

На рис. 12.1 показана діаграма станів НД і можливих переходів між ними. Перехід з одного стану в інший відбувається при виклику тих чи інших методів.

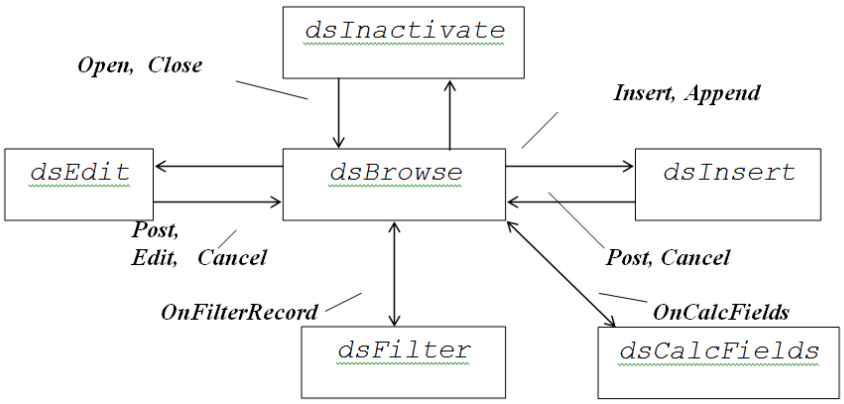


Рисунок 12.1 - Діаграма станів і переходів НД.

Це такі методи:

Open – відкриває з'єднання НД із БД (властивість ***Active*** встановлюється в *True).* Властивість ***State*** встановлюється в значення *dsBrowse;*

Close – закриває з'єднання з БД (властивість ***Active*** встановлюється в *False*). Властивість ***State*** встановлюється в значення *dsInactivate*;

Edit – НД переводиться в стан редагування. Властивість ***State*** встановлюється в значення *dsEdit*;

Insert – НД переводиться в стан вставки нового запису. Властивість ***State*** встановлюється в значення *dsInsert;*

Append – те ж, що і ***Insert***, але запис додається в кінець НД;

Post – виробляється пересилання даних поточного запису в БД.

Крім властивості ***State*** розглянемо ще такі властивості:

Active – при установці цієї властивості в *True* установлюється з'єднання з БД (викликається метод ***Open***);

DataBaseName – псевдонім БД, з яким установлюється зв'язок даного НД. Значення вибирається зі списку, що випадає;

Bof **–** приймає значення *True*, якщо поточним є перший запис (курсор установлений на перший запис);

Eof **-** приймає значення *True*, якщо курсор змістився за останній запис.

12.1.3. Навігація по НД

Поточним записом називають той запис, дані якого в даний момент доступні в прикладної програмі. З поняттям поточного запису зв'язане поняття курсору НД. *Курсор НД* указує на поточний запис.

Під навігацією по НД розуміється переміщення курсору від одного запису до іншого.

Для навігації по НД використовуються методи:

First – переміщення курсору на перший запис;

Last **–** переміщення курсору на останній запис;

Next **–** переміщення курсору на наступний запис;

Prior – переміщення курсору на попередній запис;

MoveTo(i) **-** переміщення курсору на *i* записів.

Якщо НД знаходиться в стані редагування (*dsEdit*) чи вставки нового запису (*dsInsert*), то при переміщенні до нового запису автоматично викликається метод ***Post***, що пересилає дані з поточного запису в БД, якщо у ньому були виконані зміни.

Розглянемо простий приклад навігації з програмного коду. Нехай, потрібно підрахувати кількість записів, що маються в НД. Це можна зробити шляхом послідовного "перегляду" усіх записів у НД. Фрагмент коду програми, у якому підраховується число записів у таблиці Table_Stud, має наступний вид:

```
. . . . . .
n:=0;
Table_Stud.First;
while not Table_Stud.Eof do
Begin
  n:=n+1;
  Table_Stud.Next;
End;
. . . . . .
Label1.Caption:=IntToStr(n);
. . . . . . . . . . . .
```

Для підрахунку кількості записів тут використовується змінна *n*. Мітка *Label1* використовується для відображення отриманого результату *n*. *IntToStr* – це функція, за допомогою якої виробляється перетворення цілого значення *n* у строковий тип даних. Отримане строкове представлення *n* привласнюється властивості строкового типу *Caption*.

12.1.4. Фільтрація записів у НД

У НД мається дві властивості, за допомогою яких можна включати і виключати фільтрацію. Нагадаємо, що під *фільтром* розуміється засіб, за допомогою якого можна включати в НД тільки ті записи, що задовольняють заданій умові фільтрації.

Властивості, що використовуються для фільтрації, це наступні:

Filter – властивість строкового типу, у якої задається умова фільтрації. Так само, як і в інструкції WHERE в операторі SELECT, умова фільтрації задається логічним виразом. Наприклад:

(**Fam** =‘F*’)AND(**GodRogd**>1983).

Filtered – властивість логічного типу, у якої можуть встановлюватися такі значення:

False – фільтрація виключена;

True – фільтрація включена.

FilterOptions – властивість множинного типу, у якої можна задати опції (додаткові умови) фільтрації. У властивості можуть бути задані такі константи:

foCaseInsensitive – не враховувати регістр символів;

foNoPartialCompare – перевірка на повний збіг.

Таким чином, для включення фільтра потрібно наступне:

1) у властивості ***Filter*** задати умова фільтрації. Наприклад:

Table_Stud.***Filter:***=(**Fam**=‘F*’)AND(**GodRogd** > 1983);

2) властивість ***Filtered*** установити в значення *True*. Наприклад: Table_Stud.***Filtered:****=True;*

Зазначені дії по включенню і виключенню фільтра можна робити, наприклад, в оброблювачах події натискання відповідних кнопок (подія ***OnClick***).

Для того, щоб задати умова фільтрації без обліку регістра символів, потрібно у властивості ***FilterOptions*** указати:

*Table_Stud.****FilterOptions:****=[foCaseInsensitive];*

12.2. Відмінні риси компонента TQuery

Як уже відзначалося, клас компонентів **TQuery** (як і **TTable**) породжений від класу **TDataSet**. Отже, усі розглянуті вище властивості і методи класу **TDataSet** рівною мірою доступні й у компонентах **TQuery.**

Основна відмінність компонента **TQuery** (це уже відзначалося) полягає в тому, що набір даних формується на основі оператора SELECT. Компонент **TQuery** рекомендується використовувати для доступу до віддалених БД, що працюють в архітектурі “клієнт-сервер”.

У палітрі компонентів *Delphi* компонент **TQuery** (так само, як і **TTable**) розташований на сторінці Data Access. Значок компонента має вид 

.

У компонента **TQuery** є властивість ***SQL*** – властивість текстового типу, у якої міститься текст *SQL*-запиту (оператора SELECT). Текст *SQL*-запиту можна ввести як за допомогою Інспектора Об'єктів, так і в програмі.

Розглянемо, як задати властивість ***SQL*** за допомогою Інспектора Об'єктів.

Помістити на форму компонент **TQuery** і у її властивості ***DataBaseName*** установити псевдонім БД, з якої передбачається одержувати дані. Після цього у вікні Інспектора Об'єктів потрібно виділити властивість ***SQL***. Рядок властивості має наступний вид: 

. У цьому рядку потрібно натиснути кнопку, у результаті чого відкриється вікно текстового редактора, у якому необхідно ввести текст *SQL*-запиту. Вид вікна цього редактору і приклад тексту *SQL*-запиту показаний на рис. 12.2.

Цим запитом у компонент **TQuery** будуть поміщені всі записи з таблиці СТУДЕНТИ. Після введення тексту запиту натисніть кнопку **ОК**. Текст запиту буде збережений у властивості ***SQL.*** Вікно редактора закриється.

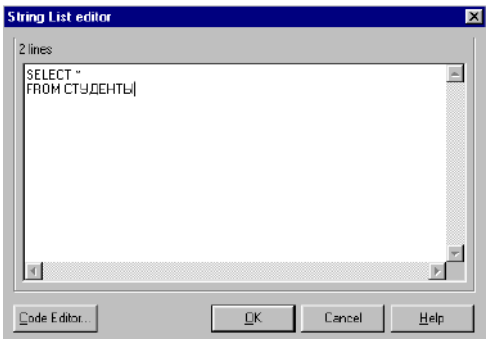


Рисунок 12.2 - Вікно редактора властивості ***SQL***

Правильність тексту запиту перевіряється тільки при відкритті набору даних, тобто при установці властивості ***Active*** у значення *True*.

Якщо при спробі установити властивість ***Active*** у значення *True* з'явиться повідомлення про помилку, варто повернутися в редактор тексту *SQL*-запиту, виправити помилки і знову спробувати установити з'єднання з БД (властивість ***Active*** установити в *True*). В іншому робота з компонентом **TQuery** аналогічна роботі з компонентом **TTable**.

12.3. Створення зв'язків між НД типу “один до кількох”

Технологія зв'язування НД **TTable** і **TQuery** реалізується по різному. Розглянемо коротко суть кожної з них. Для конкретності будемо розглядати приклад створення зв'язку між таблицями ГРУПИ (головна) і СТУДЕНТИ (підлегла).

Відзначимо також одну обов’язкову вимогу – *поля, що будуть використовуватися як ключі зв'язку, повинні бути індексованими*. Це повинно бути зроблене раніше, при створенні таблиць у СУБД *Access*.

12.3.1. Створення зв'язку “один до кількох” між НД TTable

Методика створення зв'язку між НД **TTable** наступна:

1) Закрити підлеглий НД (для цього властивість ***Active*** компонента Table_Stud установити в *False*);

2) У властивості ***MasterSource*** підлеглого НД (Table_Stud) установити ім'я джерела даних, зв'язаного з НД, що повинен бути головним (у нашому прикладі це компонент DataSource_Grupp);

3) У властивості ***MasterFields*** установити поля, що повинні бути *ключами зв'язку* в створюваному зв'язку. Для цього можна використовувати Майстер зв'язування полів, який відкривається при натисканні кнопки у рядку властивості ***MasterFields***: 

. Вікно Майстра зв'язування полів має вид, показаний на рис. 12.3.

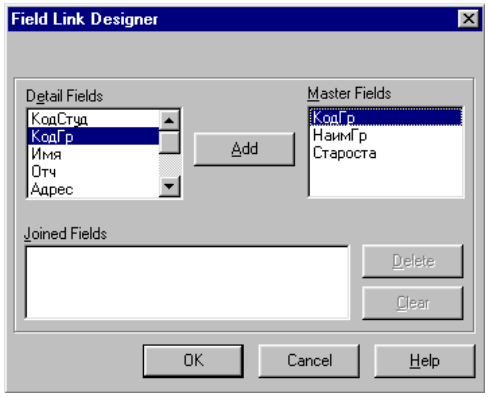


Рисунок 12.3 - Майстер зв'язування полів (вибір полів зв'язку)

У списках полів підлеглого НД (ліворуч) і головного НД (праворуч) виділити (мишею) поля, по яких повинен встановлюватися зв'язок між НД. У розглянутому прикладі це однойменні поля **КодГр**. Після їх виділення натисніть кнопку **Add**. В області Joined Fields (зв'язані поля) з'явиться зображення створюваного зв'язку, як це показано на рис. 12.4.

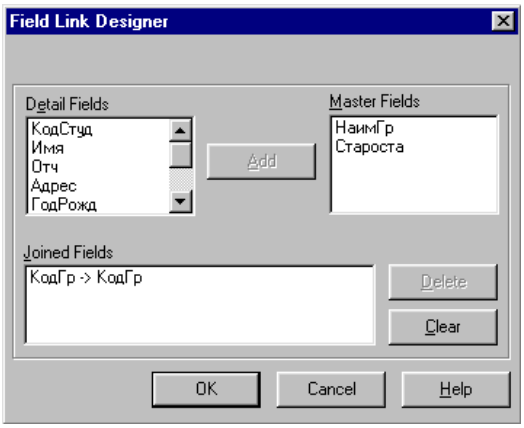


Рисунок 12.4 - Майстер зв'язування полів (поля зв'язку обрані)

У вікні Інспектора Об'єктів у властивості ***MasterFields*** з'явиться ім'я поля зв'язку головного НД **КодГр**. Одночасно у властивості ***IndexFieldNames*** з'явиться ім'я індексованого поля підлеглого НД (у нашому прикладі – це так само **КодГр**);

4) Відновити з'єднання підлеглого НД із БД (властивість ***Active*** підлеглого НД Table_Stud установити в *True*).

Зв'язок між НД Table_Grupp (головний) і НД Table_Stud (підлеглий) створений.

Тепер можна перевірити роботу створеного зв'язку в такий спосіб. При виборі якого-небудь запису в сітці DBGrid_Grupp, що зв'язана з головним НД, у сітці DBGrid_Stud, яка зв'язана з підлеглим НД Table_Stud, будуть відображатися тільки ті записи, що відносяться до обраної групи.

12.3.2. Створення зв'язку “один до кількох” між НД TQuery

Так само будемо розглядати приклад, коли є створені на основі компонентів **TQuery** НД Query_Grupp і Query_Stud, у яких представлена інформація відповідно з таблиць ГРУПИ і СТУДЕНТИ. Створення зв'язку “головний-підлеглий” між цими НД у цьому випадку виробляється в наступній послідовності:

1) Закрити підлеглий НД (властивість ***Active*** компонента Query_Stud установити в *False*);

2) Викликати текстовий редактор *SQL*-запиту (так само, як розглянуто вище) і ввести такий текст:

```
SELECT *
FROM СТУДЕНТИ
WHERE  КодГр =: КодГр
```

Додана інструкція WHERE, у якій задана умова добору по рівності значень поля **КодГр** і значення *параметра*, що заданий ім'ям після двокрапки (після “**=:**”). Як параметр необхідно задати ім'я поля – ключа зв'язку головного НД. Задаємо – **КодГр**.

3) У властивості ***DataSource*** необхідно установити ім'я джерела даних, з якого повинні виходити поточні значення параметра. У нашому прикладі у властивості ***DataSource*** установимо значення DataSource_Grupp – ім’я компонента – джерела даних, що зв’язаний з головним НД. Завдяки цьому як параметр у *SQL*-запит буде передаватися значення поля **КодГр** із НД Query_Grupp, що відповідає значенню зі поточного запису (обраного в сітці BGrid_Grupp).

4) Відкрити підлеглий НД Query_Stud (його властивість ***Active*** установити в значення *True*).

Зв'язок “головний-підлеглий” між НД Query_Grupp і Query_Stud встановлено.

Переконатися в “роботі” зв'язку можна так само, як і раніше: у сітці DBGrid_Stud повинні відображатися тільки ті дані, що відповідають поточній групі, обраної в даний момент у сітці головного НД DBGrid_Grupp.

12.4. Об'єкти-поля в наборах даних

12.4.1. Клас об'єктів-полів TField

Набір даних (НД) складається з записів, а записи складаються з полів. Окремі поля представляються об'єктами, породженими від класу **TField.**

Клас **TField** є абстрактним і безпосередньо не використовується. Використовуються його типізовані нащадки. На рис. 12.5 показаний фрагмент ієрархії класів і деякі з типізованих класів-полів.

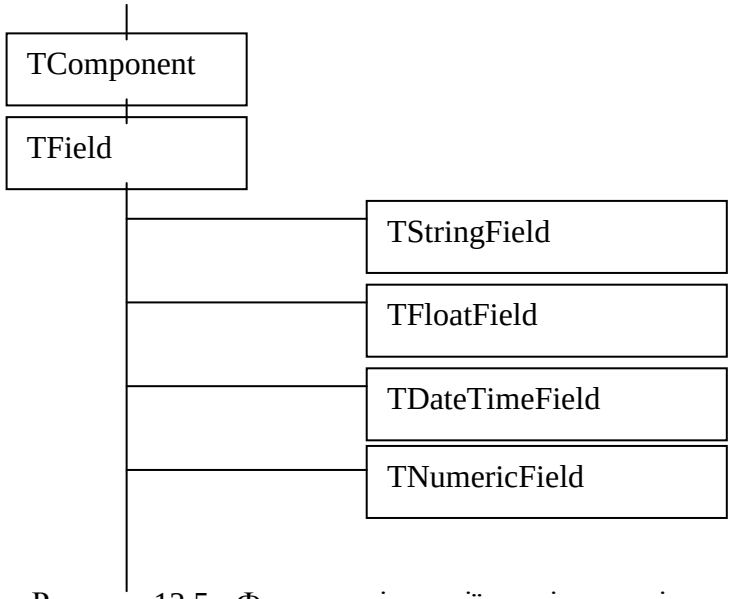


Рисунок 12.5 - Фрагмент ієрархії класів, що містить клас **TField**

Об'єкти, породжені від типізованих класів – нащадків **TField**, є не візуальними компонентами, призначеними для доступу до даних, що зберігаються в окремих полях, і для керування цими даними.

При відкритті набору даних *автоматично* створюються об'єкти-поля для кожного поля НД. Властивості всіх об'єктів-полів установлюються відповідно до типів даних, заданих у таблицях, на основі яких створюється цей НД.

У НД є властивість ***Fields***, що є масивом, елементами якого є об'єкти-поля. Звернутися до значення якого-небудь поля можна через властивість ***Value***, наприклад, у такий спосіб:

Table_Stud.*Fields*[*i*].*Value* ,

де *i* – індекс (порядковий номер) даного поля.

Інший спосіб звертання до даних (більш зручний) – звертання по імені за допомогою властивості ***FieldByName***. Наприклад, до поля **КодСтуд** можна звернутися в такий спосіб:

Table_Stud.FieldByName(‘КодСтуд’).Value

Властивість ***Value*** містить дані в тім форматі, що відповідає типу даних поля. Для перетворення даних до потрібного типу можна використовувати спеціальні властивості, наприклад:

AsString – перетворення до строкового типу;

AsInteger – перетворення до цілого типу;

AsFloat – перетворення до речовинного типу.

Наприклад, для відображення даних поля **КодСтуд** (цілий тип) за допомогою компонента **TLabel** можна застосувати такий оператор:

Label1.***Caption***:=Table_Stud.***FieldByName***(**‘КодСтуд’**).***AsString***;

12.4.2. Редагування полів у НД

Як уже відзначалося, при розміщенні на формі компонента НД в нього за замовчуванням включаються всі поля, що маються в таблиці. Однак, часто буває потрібно змінити склад полів (видалити які-небудь поля, додати нові). Для цього зручно використовувати Редактор Полів.

Викликати Редактор Полів можна подвійним щигликом на значку компонента НД (**TTable** чи **TQuery**) чи за допомогою контекстного меню (клацнути на значку компонента правою кнопкою миші і вибрати команду Fields Editor). При першому відкритті Редактора Полів вікно його порожнє (рис. 12.6). Це означає (за замовчуванням), що в НД включені всі поля.

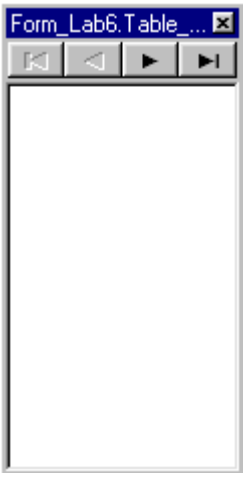


Рисунок 12.6 - Вікно Редактора Полів при першому його відкритті

Для додавання потрібних полів необхідно викликати контекстне меню Редактора Полів (подвійним клацанням миші в будь-якому місці вікна Редактора Полів) і виконати команду Add Fields. Відкриється вікно додавання полів, у якому потрібно виділити поля, що додаються, і натиснути кнопку **OK**. Якщо потрібно виділити одночасно кілька полів, то виділення робити при натиснутій клавіші «***Ctrl*** ». У вікні Редактора Полів відобразяться всі обрані поля.

На рис. 12.7 показаний вид вікна, у якому відображаються всі поля з НД Table_Grupp.

Рисунок 12.7 - Редактор Полів НД Table_Grupp

При необхідності можна змінити послідовність розташування полів у НД шляхом їхнього перетаскування мишею у вікні Редактора Полів.

При виділенні якого-небудь поля в Редакторі Полів в Інспекторі Об'єктів відображаються властивості виділеного поля. Їх можна встановлювати за бажанням розроблювача програми. Наприклад, під час лабораторних робіт ми будемо використовувати і встановлювати значення для таких властивостей:

Alignment – вирівнювання розташування даних: уліво, вправо, по центрі;

DisplayLabel – заголовок стовпця в таблиці;

DisplayWidth – ширина стовпчика (число символів);

ReadOnly – тільки для читання.

За допомогою Редактора Полів легко створювати нові поля. Як це робити – розглянемо далі.

12.4.3. Створення поля, що обчислюється

Нехай, наприклад, у НД Table_Stud ми хочемо створити поле, що обчислюється, "Вік". Створення поля виробляється в наступній послідовності.

1) Викликати Редактор Полів (подвійним клацанням на компоненті Table_Stud) і додати в ньому потрібні поля.

2) Виконати команду контекстного меню Редактора Полів New Field. У результаті цього відкриється вікно Майстра створення полів (рис. 12.8).

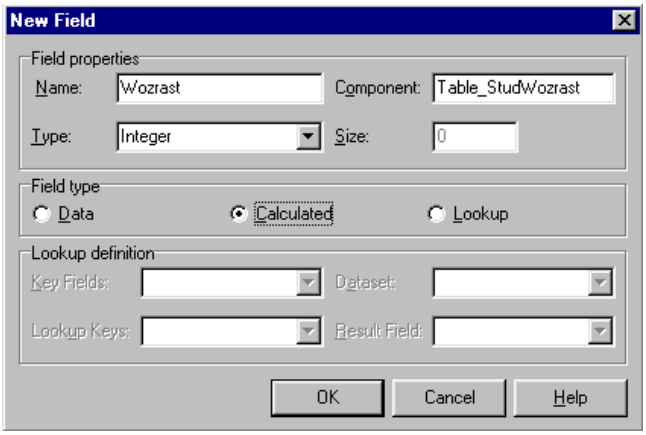


Рисунок 12.8 - Створення поля, що обчислюється

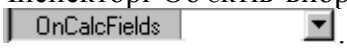
3) В області Field properties (Властивості поля) увести такі значення властивостей:

Name – **Wozrast**

Type – Integer (шляхом вибору зі списку).

Властивість ***Size*** (Розмір) уводиться тільки для строкового типу поля. Властивість ***Component*** (Ім'я об'єкта-поля) встановлюється автоматично.

4) В області Field Type (Тип поля) задати тип поля: для поля, що обчислюється, вибрати **Calculated** (Показане на Рис. 12.8). Після цього натисніть кнопку **ОК**. Вікно Майстра закриється.

5) Створити оброблювач події ***OnCalcFields*** (для НД Table_Stud). Для цього виділити компонент Table_Stud на формі й в Інспекторі Об'єктів вибрати сторінку Events і виділити на ній властивість-подію ***OnCalcFields*** 

. В області значення цієї властивості зробити подвійне клацання мишею. У результаті відкриється вікно редактора коду модуля, у якому буде зроблена заготівка тексту для оброблювача події. Між дужками **Begin** і **end** потрібно ввести такий текст:

Table_Stud.**Wozrast.*AsInteger*:=2003-**Table_Stud.***FieldByName*(**'ГодРожд').*AsInteger*;**

Цей оператор буде виконуватися щоразу, коли в НД буде відбуватися перехід до нового запису. При цьому властивість ***AutoCalcFields*** НД повинна бути встановлена в *True*. (це значення встановлюється за замовчуванням).

12.4.4. Створення поля підстановки

Так само будемо розглядати приклад, коли в НД Table_Stud потрібно створити поле підстановки **NameGr** (Найменування групи), значення якого буде "підставлятися" із НД Table_Grupp.

Для створення поля підстановки потрібно виконати наступне:

1) Викликати для НД Table_Stud Редактор Полів (як і раніше).

2) Викликати Майстер створення полів (командою контекстного меню New Fields. Відкриється вікно Майстра (Рис. 12.9).

3) В області Field properties (Властивості поля) увести такі значення властивостей:

Name – **NameGrupp**

Type – String (шляхом вибору зі списку).

Size – 10

Component – Table_StudNameGrupp (встановлюється автоматично).

4) В області Field Type (Тип поля) задати тип поля: для поля підстановки вибрати опцію Lookup (рис. 12.9).

5) В області Lookup definition (Визначення підстановки) установити такі дані (вибором потрібного значення зі списку):

Key Fields – установити ім'я поля – ключа зв'язку НД Table_Stud (установити **КодГр**);

DataSet – установити ім'я НД, з якого буде виконуватися підстановка даних (установити Table_Grupp);

Lookup Keys – установити ключове поле НД Table_Grupp (установити **КодГр**);

Result Field – установити ім'я поля, дані якого повинні підставлятися в створюване поле підстановки (установити **НаимГр**).

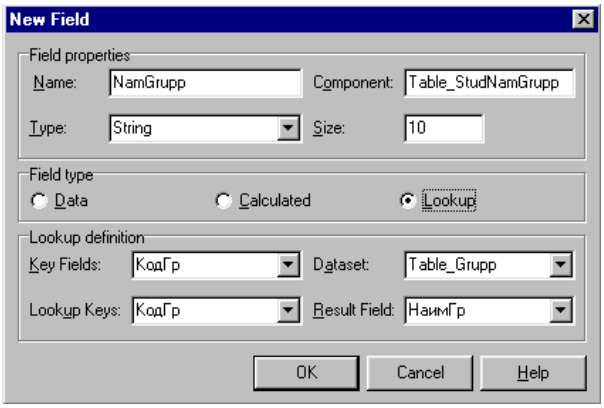


Рисунок 12.9 - Створення поля підстановки.

Після цього натисніть кнопку **ОК**. Поле підстановки створене. У вікні Редактора Полів з'явиться створене поле **NameGrupp**.

Для створеного поля можна задати підпис (заголовок стовпця). У прикладі, що розглядався, доцільно створити підпис “Група”. Для цього в Редакторі Полів виділите поле **NameGrupp**, і потім в Інспекторі Об'єктів введіть у властивості ***DisplayLabel*** (Підпис) значення **Група.**

12.5. Компоненти відображення даних

12.5.1. Компонент TDBGrid

На третьому етапі створення додатка баз даних необхідно розробити користувальницький інтерфейс на основі компонентів відображення даних. Ці компоненти призначені спеціально для рішення задач перегляду і редагування даних. Зовні більшість цих компонентів нічим не відрізняються від стандартних елементів керування. Більш того, багато хто з компонентів відображення даних є спадкоємцями стандартних компонентів — елементів керування.

Компоненти відображення даних повинні бути зв'язані з компонентом TDataSource і через нього з компонентом набору даних. Для цього використовується їхня властивість DataSource. Воно присутнє у всіх компонентах відображення даних.

Більшість компонентів призначені для представлення даних з одного єдиного поля. У таких компонентах мається ще одна властивість DataField, що визначає поле зв'язаного набору даних, відображуване в компоненті.

Особливе значення для додатків баз даних грає компонент TDBGrid, що представляє дані у виді таблиці. У стовпцях таблиці розміщаються поля набору даних, а в рядках — запису. Для цього компонента не має змісту визначати конкретне поле, але можна задати набір стовпчиків, що набудовується, а для кожної з них визначити поле набору даних.

Компонент **TDBGrid** являє собою двовимірну таблицю (сітку), у якій рядки відповідають записам НД, а стовпці відповідають окремим полям кожного з записів.

Компонент **TDBGrid** має дуже потужні засоби для роботи з даними. Наприклад, із сітки (компонента) можна здійснювати навігацію по НД, редагувати, видаляти, вводити нові записи. Вид компонента в режимі виконання прикладної програми показаний на рис. 12.10.

КодСтуд	КодГр	Фам	Имя	Отч	Адрес
1	1	Иванов	Иван	Иванович	г. Киев
2	4	Петров	Петр	Петрович	г. Киев
3	1	Иващенко	Олег	Николаевич	г. Бровары
4	2	Тоценко	Виталий	Георгиевич	г. Киев
5	1	Атрощенко	Петр	Леонидович	г. Киев
6	2	Иванченко	Василий	Семенович	г. Клавдиево

Рисунок 12.10 - Вид компонента **TDBGrid** при працюючої програмі

У лівій частині сітки розташована *область індикації*, у якій розташовується покажчик (курсор) поточного запису. Якщо запис знаходиться в стані редагування, то покажчик приймає вид. Новий (що додається) запис позначається покажчиком.

Число записів (рядків), розміщених у компоненті **TDBGrid,** змінити не можна. У ньому завжди містяться всі записи зі зв'язаного НД. Кількість стовпців, відображуваних у компоненті, їхній вид і черговість можна змінювати.

Однією із властивостей компонента є властивість ***Columns*** – список об'єктів-стовпців, включених у НД. За замовчуванням у властивість ***Columns*** включаються всі стовпці зі зв'язаного НД. За бажанням розроблювача властивість ***Columns***, а також властивості окремих об'єктів-стовпців (ширина, заголовок, шрифт і т.д.) можна змінювати. Це можна робити за допомогою Редактора Стовпців.

Викликати Редактор Стовпців можна одним з наступних способів:

- подвійним клацанням на компоненті **TDBGrid**;
- командою Columns Editor контекстного меню компонента;
- натисканням кнопки в рядку Columns ridColumns) властивості ***Columns*** в Інспекторі Об'єктів.

При першому відкритті Редактора Стовпців відкривається порожнє вікно Редактора, вид якого показаний на рис. 12.11.

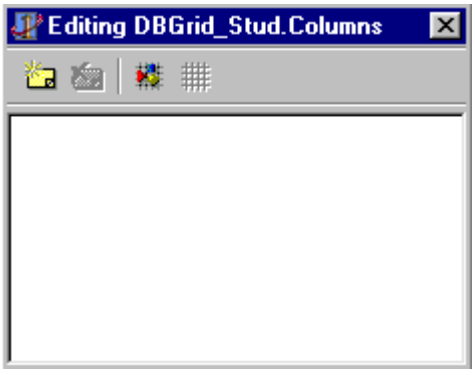


Рисунок 12.11 - Порожнє вікно Редактора Стовпців

Якщо натиснути на панелі інструментів кнопку Add All Fields  (Додати всі поля), у Редактор Стовпців будуть додані стовпці, що відповідають усім полям зі зв'язаного НД. На рис. 12.12 показане вікно Редактора Стовпців, коли в Редактор додані всі стовпці з НД Table_Stud.

За допомогою кнопок на Панелі Інструментів Редактора Стовпців (чи за допомогою команд контекстного меню) можна видаляти, додавати стовпці, редагувати їхні властивості в Інспекторі Об'єктів.

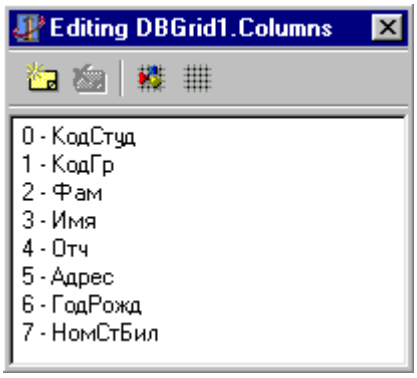


Рисунок 12.12 - У Редактор Стовпців додані всі стовпці з НД Table_Stud

Якщо виділити мишею який-небудь стовпець, то у вікні Інспектора Об'єктів відобразяться властивості цього стовпця, які там же можна редагувати. Під час лабораторних занять ви будете редагувати такі властивості:

Font – шрифт;
Color – колір;
Title – властивості заголовка стовпця;
Width – ширина у пікселях, і ін.

12.5.2. Компонент TDBNavigator

Компонент **TDBNavigator** у Палітрі компонентів розташований на сторінці Data Controls. Його вид після розміщення на формі показаний на рис. 12.13.

Рисунок 12.13 - Компонент **TDBNavigator**

Компонент являє собою набір кнопок, кожна з який виконує визначені дії над НД. Призначення кнопок наступне:

 - перший запис (nbFirst);
 - попередній запис (nbPrior);
 - наступний запис (nbNext);
 - останній запис (nbLast);
 - додати запис (nbInsert);
 - видалити запис (nbDelete);
 - редагувати запис (nbEdit);
 - зберегти зміни (nbPost);
 - скасувати зміни (nbCancel);
 - обновити дані (nbRefresh).

При розробці прикладної програми в компонент можна включати будь-яку кількість кнопок у будь-якому їхньому сполученні.

Включенням кнопок у компонент можна керувати за допомогою властивості ***VisibleButtons***, що є множиною, яка містить константи, що представляють різні кнопки. Імена відповідних кнопкам констант були зазначені вище у дужках.

Для зв'язування компонента **TDBNavigator** із НД, яким він повинний керувати, служить властивість ***DataSource***. У цій властивості потрібно установити (вибором зі списку) ім'я джерела даних, зв'язаного з потрібним НД.

Коли в прикладної програмі ви використовуєте сітку **TDBGrid**, то, кажучи взагалі, необхідності в **TDBNavigator** немає, тому що всі передбачені в ньому дії реалізовані відповідними методами в компоненті **TDBGrid**.

12.5.3. Компоненти TDBText і TDBEdit

Компонент **TDBText** являє собою статичний текст, що відображає поточне значення деякого одного поля зі зв'язаного НД. Дані в компоненті відображаються в режимі “тільки для читання”, тобто редагувати дані не можна.

Компонент **TDBEdit** дозволяє не тільки переглядати, але і редагувати дані поля. У **TDBEdit** можна використовувати буфер обміну.

Для зв'язування обох компонентів із НД використовуються властивості:

DataSource – компонент - джерело даних, зв'язаний з потрібним НД;

DataField – ім'я поля, з яким повинний бути зв'язаний компонент.

Контрольні запитання:

1. Поняття набору даних. Компоненти, що є наборами даних, найбільш важливі їх властивості.
2. Стан набору даних.
3. Навігація по набору даних.
4. Фільтрація записів в НД.
5. Відмінні особливості і властивості компонента **TQuery.**
6. Створення між набраними даними зв’язку типу “головний-підлеглий”. В чому суть цього зв’язку?

7.Поняття об’єктів-полів. Клас **TField** і його властивості.

8.Як редагувати поля в НД?

9.Створення полів, що обчислюються.

10.Створення полів підстановки.

11.Компонент **TDBNavigator**: призначення, основні властивості.

12.Компоненти **TDBText** і **TDBEdit**: призначення та найбільш важливі властивості.

13. Архітектура “клієнт-сервер”. Сервер БД

13.1. Загальні поняття

Найбільш цікавим для практики є використання БД у мережі, коли кілька користувачів одночасно повинні працювати з даними однієї і тієї ж БД.

Раніше ми уже відзначали, що, взагалі кажучи, локальну БД, наприклад, створену в *Access*, можна установити на мережному сервері і дозволити користувачам працювати з цією БД через мережу. Усе буде добре доти, поки з БД буде працювати невелике число користувачів - клієнтів. Якщо число одночасно працюючих користувачів зросте (наприклад, 10 і більш), то відразу ж виявляться недоліки такої архітектури БД (її називають архітектурою «файловий сервер») – різко знижується продуктивність інформаційної системи, відбувається уповільнення роботи системи, зростає час її реакції. При значному зростанні числа клієнтів (100 і більше) інформаційна система з такою архітектурою взагалі стає непрацездатною. Основні причини цих недоліків очевидні і полягають у наступному:

- По мережі передається великий обсяг “зайвої” інформації, тому що обмін даними відбувається шляхом пересилання файлів, хоча клієнту може бути потрібна тільки невелика частина інформації з файлу. При збільшенні числа клієнтів швидко настає перевантаження каналів зв'язку;
- Низька захищеність даних від внесення в них помилок через те, що обробка даних виконується кожним клієнтом незалежно, на своїх ПК. Нагромадження в БД помилкових і суперечливих даних згодом приводить до її руйнування і непрацездатності;
- Недостатня захищеність даних від несанкціонованого доступу – захист обмежений тільки засобами, що є на мережному сервері.

Тому розроблювачі СУБД, призначених для роботи великої кількості користувачів, неминуче прийшли до необхідності побудови нової архітектури БД, у якій би вирішувалися зазначені вище проблеми. Ця нова архітектура була названа **архітектурою “клієнт-сервер”**. Ця архітектура реалізована в сучасних версіях промислових СУБД, наприклад, таких, як *MS SQL Server, Oracle, Informix, InterBase* і багатьох інших.

Найбільш характерні ознаки (особливості) архітектури “клієнт-сервер” наступні:

- Клієнту пересилаються тільки ті дані, що були їм запитані. Завдяки цьому істотно *скоротилося непродуктивне завантаження каналів зв'язку* (мережі) і, отже, з'явилася можливість збільшити кількість одночасно працюючих клієнтів;
- Основна обробка даних перенесена на комп'ютер-сервер, на якому встановлена БД. Завдяки цьому *зменшилося до мінімуму кількість помилок і протиріч*, внесених у дані. Крім того, з'явилася можливість реалізувати більш стійкий *захист від несанкціонованого доступу* до даних, що є винятково важливим для інформаційних систем з великою кількістю користувачів;
- Розроблені і реалізовані нові технологічні ідеї, завдяки яким вдалося створити дуже ефективні алгоритми керування процесом роботи великої кількості клієнтів. Однією з таких ідей є *використання механізму транзакцій*.

Суть усіх проблем, що виникають при багатокористувацькій роботі з даними, можна пояснити наступним прикладом.

Нехай, наприклад, ви один із клієнтів, що працюють із загальної БД, і в деякий момент часу ви прочитали який-небудь запис і робите його редагування. У цей же час інший клієнт також звернувся до цього ж запису і намагається змінити в ньому дані чи взагалі видаляє його. Що і як буде відбуватися після того, як ви спробуєте зберегти зроблені зміни? Ясно, що виникає конфліктна ситуація! Розв’язання подібних конфліктів здійснюється на основі спільного застосування механізмів транзакцій і блокувань.

13.2. Поняття транзакції

Транзакцією називають групу логічно зв'язаних послідовно виконуваних операцій (операторів *SQL*), таких, що тільки успішне виконання всіх операцій приводить до реальних змін даних у БД, які є результатом виконання операцій, включеним у дану транзакцію. Якщо хоча б одна з операцій не буде успішно виконана (через збій чи з якої-небудь іншої причини), то всі зміни, що до цього часу вже були зроблені, **скасовуються**. Скасування зроблених змін називають *відкатом транзакції* (*RollBack*).

Отже, результатом транзакції може бути або фіксація в БД усіх змін, що виконувалися в даній транзакції, або незмінний колишній стан БД, у якому вона знаходилася на момент часу перед початком транзакції.

Класичним прикладом транзакції, який звичайно приводять у літературі по БД, є бухгалтерська проводка. Нехай, наприклад, у банку є рахунки клієнтів *А* и *Б*, і потрібно деяку суму *S* зняти з рахунка *А* и перевести її на рахунок *Б*. Таким чином, у даному прикладі транзакція складається з двох операцій:

- зняти суму *S* з рахунка *А*;
- помістити суму *S* на рахунок *Б*.

Ясно, що якщо одна з цих операцій буде виконана, а інша – ні, то цілісність даних буде порушена, що є неприпустимим. Завдяки механізму транзакцій гарантується, що будуть виконані обов'язково обидві операції, чи жодна з них не буде виконана. В обох випадках забезпечується збереження цілісністі даних.

При невдалому виконанні транзакції (при її відкаті), вона може бути виконана повторно.

13.3. Проблеми доступу до даних багатьох користувачів

Розглянемо тепер проблеми, що виникають при одночасному доступі до даних багатьох користувачів. Цим проблемам уже придумали спеціальні назви:

- «читання незафіксованих змін»;
- «неповторювальне читання»;
- «незафіксовані (чи фантомні) записи»;
- «загублені зміни», і ін.

Суть цих проблем коротко полягає в наступному:

«Читання незафіксованих змін». Нехай, наприклад, паралельно (одночасно) виконуються дві транзакції: *T1* і *T2*. Транзакція *T1* зробила зміни даних у запису *A*, але ще їх не зафіксувала (транзакція ще не завершилася). Транзакція *T2* прочитала зміни даних у запису *A* и продовжує виконуватися далі. Якщо тепер з якої-небудь причини відбудеться скасування транзакції *T1*, то в цьому випадку транзакція *T2* буде працювати з невірними даними.

«Неповторювальне читання». Транзакція *T1* прочитала запис *A* і продовжує з нею працювати. Транзакція *T2* до завершення *T1* змінила дані в записі *A* и зафіксувала їх у БД. При повторному читанні транзакцією *T1* даних вони можуть відрізнятися від колишніх і результати виконання транзакції *T1* можуть виявитися помилковими.

«Незафіксовані (фантомні) записи». Нехай транзакція *T1* створила новий запис *A*, але ще не зафіксувала його в БД. У цей час транзакція *T2* прочитала запис *A* и продовжує з ним працювати. Якщо транзакція *T1* буде скасована, то виявиться, що транзакція *T2* працює з неіснуючим записом.

Подібна ж ситуація відбудеться у випадку, якщо транзакція прочитала і працює з записом *A*, і після цього транзакція *T2* видалить цей запис. Транзакція *T1* буде продовжувати працювати з неіснуючим записом.

«Загублені зміни». Транзакція *T1* зробила зміни в запису *A*, але ще їх не зафіксувала. У цей час транзакція *T2* ще раз змінила дані в *A* и успішно завершилася. Зміни, зроблені транзакцією *T1*, будуть загублені.

Уже цього переліку проблемних ситуацій досить, щоб оцінити серйозність виникаючих проблем.

Основним способом дозволу цих проблем, що приводять до конфліктів, є блокування, застосовувані разом із уведенням різних рівнів ізоляції транзакцій. Про це більш детально поговоримо нижче.

13.3. Блокування

Блокування, як уже відзначалося, є основним засобом розв'язання конфліктів.

При виникненні конфлікту у транзакції є два вибори – або негайно збудити виключення (помилку), або почекати якийсь час і після цього знову спробувати виконати потрібну дію. У зв'язку з цим введене поняття режиму блокування, що може мати два значення: ***wait*** (чекати), і ***nowait*** (не чекати).

Конфлікти, про які мова йде, можуть виникати як при читанні даних, так і при введенні чи редагуванні даних.

На конфлікти при читанні даних крім режиму ***wait***/***nowait***, впливає також установлений рівень ізоляції транзакцій. На конфлікти ж при введенні чи редагуванні даних рівень ізоляції транзакцій не впливає. Тому зручно розглянути, як застосовуються блокування при вирішенні конфліктів при введенні і редагуванні даних.

13.3.1. Конфлікт при вставці запису

Нехай транзакція *T1* додає запис у таблицю, але ще не підтвердила зміни. У цей час транзакція *T2* так само намагається вставити запис з таким же значенням первинного ключа (рис. 13.1).

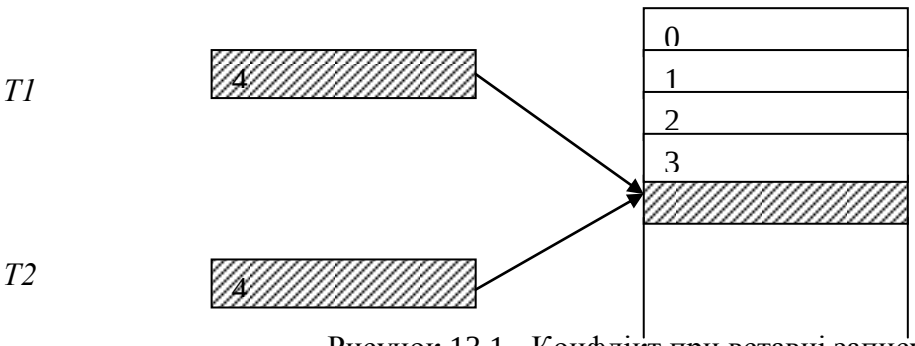


Рисунок 13.1 - Конфлікт при вставці запису

У залежності від режиму блокування транзакції *T2* можливі такі варіанти:

1) якщо *T2* запущена в режимі ***wait***, то вона чекає завершення *T1*. Якщо *T1* завершується підтвердженням, то для *T2* генерується виключення (помилка). Якщо *T1* не підтверджується (зроблений відкат *T1*), то *T2* успішно завершується;

2) якщо *T2* запущена в режимі ***nowait***, то для *T2* відразу генерується помилка.

13.3.2. Конфлікт при редагуванні запису

Транзакція *T1* змінила дні у якому-небудь запису, але ще не підтвердила зміни. Транзакція *T2* намагається редагувати чи видалити цей самий запис. У залежності від режиму блокування для *T2* можливі такі варіанти:

1) якщо *T2* запущена в режимі ***wait***, то вона буде чекати, поки *T1* не підтвердиться чи скасується. Якщо *T1* підтвердилася, то для *T2* генерується помилка. Якщо *T1* не підтвердилася, то *T2* успішно завершується;

2) якщо *T2* запущена в режимі ***nowait***, то відразу генерується помилка.

13.3.3. Взаємне блокування транзакцій

Нехай запущені дві трансакції: *T1* і *T2*, і є два запису: *А* и *Б*. Припустимо, виконується така послідовність дій:

- Транзакція *T1* заблокувала запис *А* і успішно з ним працює;
- Транзакція *T2* заблокувала запис *Б* и також успішно з ним працює;
- Транзакції *T1* знадобилося попрацювати з записом *Б* и вона намагається його заблокувати. Тому що запис *Б* зайнятий, *T1* переходить у режим чекання;
- Тепер транзакції *T2* треба було попрацювати з записом *А*. Вона намагається його заблокувати і також переходить у режим чекання.

У результаті виникла сумна ситуація: обидві транзакції не мають ніякої можливості продовжити своє виконання, тому що “намертво” заблокували одна одну.

На будь-якому сервері, у тому числі і *InterBase*, маються способи вийти з такої ситуації. Сервер періодично перевіряє стан усіх запущених транзакцій і при виявленні випадку взаємного блокування одну з транзакцій відкачує примусово.

13.4. Рівні ізоляції транзакцій

Завдяки механізму транзакцій можна зробити цілком “невидимими” для всіх інших транзакцій (користувачів) зміни, вироблені в даної транзакції аж до її підтвердження. Однак така повна ізольованість транзакцій не завжди ефективна, тому що приводить до уповільнення роботи інформаційної системи. Продуктивність системи може бути збільшена, якщо дозволити транзакції “бачити” ті зміни, що були підтверджені іншими транзакціями вже після запуску даної транзакції.

Для більш гнучкого керування транзакціями було введене поняття рівня ізоляції транзакцій, що визначає, які зміни, зроблені в інших транзакціях, будуть видні в даної транзакції.

Існують чотири рівні ізоляції транзакцій:

DIRTY READ. Транзакція з таким рівнем ізоляції може читати непідтверджені дані в інших транзакціях. Цей рівень називають “брудне читання”. У *InterBase* такий рівень ізоляції транзакцій заборонений.

READ COMMITED. Транзакція, запущена з таким рівнем, може читати зміни, зафіксовані в паралельно виконуваних транзакціях. Цей рівень гарантує, що ми не зможемо прочитати непідтверджені змінені дані в інших транзакціях, але підтверджені дані прочитати можна.

SNAPSHOT. Цей рівень ізоляції використовується для створення “моментального знімка” БД у момент запуску транзакції. Всі операції читання даних у цієї транзакції будуть “бачити” БД у стані, у якому вона була в момент запуску транзакції. Усі зміни в інших транзакціях (підтверджені і непідтверджені) не видимі для цієї транзакції.

SNAPSHOT TABLE STABILITY. Цей рівень ізоляції аналогічний рівню **SNAPSHOT**, але додатково блокує таблицю на запис. Ідея тут проста – якщо транзакція з таким рівнем ізоляції робить зміни в якій-небудь таблиці, то транзакції з рівнями **READ COMMITTED** і **SNAPSHOT** можуть тільки читати цю таблицю, а транзакції з таким же рівнем ізоляції, тобто **SNAPSHOT TABLE STABILITY**, не можуть і читати її дані.

Цей рівень ізоляції рекомендується використовувати в коротких за часом оновлюючих транзакціях.

14. Сервер InterBase SQL Server

14.1 Призначення та загальні відомості

Сервер InterBase SQL Server призначений для збереження й обробки великих обсягів інформації при одночасній роботі з БД багатьох клієнтів. Узагалі говорячи, таке ж призначення має і будь-який інший *SQL*-сервер, однак *InterBase* має ряд відмінних рис, малий, швидкий, високо ефективний завдяки яким інтерес до *InterBase* у світі зараз дуже великий. Потрібна база даних, яка є компактною в пам'яті та розмірі диска, але достатньо потужна, щоб надійно підтримувати ваші важливі для бізнесу програми? InterBase легко вставляти або встановлювати, має крихітний слід, підтримує потужне шифрування та вимагає нульового адміністрування. Це неймовірна база даних для вбудовування в додатки.

InterBase є кросплатформеним продуктом, який сумісний з різними типами операційних систем, включаючи *Windows, Linux,* MacOC, Android і ін. Інсталюється на архітектури ПК x86 (32p) та x64(64p).

InterBase відрізняється надзвичайно низькими системними вимогами і при цьому високою продуктивністю і легкістю адміністрування.

Однієї з найважливіших особливостей *InterBase* є його *версійна архітектура*, що забезпечує унікальні можливості при роботі великої кількості клієнтів – пишучі користувачі ніколи не блокують читаючих користувачів. Крім того, завдяки версійній архітектурі забезпечується дуже висока надійність і стійкість роботи з даними.

У *InterBase* реалізований механізм блокувань на рівні запису. Це значить, що сервер блокує тільки ті записи, що реально були змінені користувачем, але не блокує всю сторінку даних цілком. Завдяки цьому ще більше знижується імовірність конфліктів при паралельної роботі багатьох користувачів.

InterBase був розроблений на початку 80-х років американською компанією *DEC*. Пізніше розробка *InterBase* велася фірмою *InterBase Software*, що згодом ввійшла до складу компанії *Borland* , зараз власник програми компанія Embarcadero.

В даний час застосовуються версії програмного забезпечення *InterBase 2017.* Сервер InterBase 2017 доступний в 4-х варіантах:

INTERBASE 2017 SERVER EDITION
INTERBASE 2017 DESKTOP EDITION
INTERBASE 2017 DEVELOPER EDITION
INTERBASE 2017 TOGO EDITION

Новина:

- Максимальна продуктивність БД з великою кількістю користувачів.
- Високий ступінь захисту.
- Швидке відновлення після збоїв.
- Низькі апаратні вимоги.
- Вбудовані засоби шифрування.
- Нова команда перепідключення (Reconnect).
- Новий параметр командного рядка (names).

- Моніторинг всіх онлайн баз на сервері.
- Підтримка таблиць SQL.

На основі версії *InterBase 2009*, яка є відкритим, вільно розповсюджуваним продуктом, розроблено кілька клонів, наприклад, *Firebird 2.5*.

InterBase дуже зручний для навчальних цілей, тому що його локальна версія поширюється з пакетом *Delphi XE*. Крім того, у всіх серверів БД багато загального в принципах організації, тому знайомство з *InterBase* дасть також загальні представлення і про інші, більш складних в експлуатації *SQL*-серверах.

14.2 Склад сервера InterBase

Сервер *InterBase*, як і будь-яка клієнт-серверна система керування базою даних, складається з двох частин: клієнтської та серверної. Для роботи з *InterBase* на кожній машині повинний бути встановлений клієнт *InterBase*. Основною його частиною є бібліотека Gds32.dll. На сервері БД встановлюється і клієнт, і сервер *InterBase*. Основний модуль сервера *InterBase* - файл ibserver.exe.

Всі об'єкти бази даних *InterBase* зберігаються в одному файлі. Файли баз даних *InterBase* мають розширення .GDB. Файли резервних копій баз даних мають розширення .GBK.

Файл бази даних *InterBase* має сторінкову організацію. Це означає, що дані у файлі БД фізично розділені на сторінки, розмір яких визначається при створенні бази даних. За замовчуванням розмір сторінки дорівнює 1 КБайт. Якщо таблиці БД містять більш 200 тис. записів, є сенс збільшити розмір сторінки з метою збільшення продуктивності. Як правило, розмір сторінки повинний бути дорівнювати розміру кластера жорсткого диска, на якому розміщений файл БД. Для файлових систем FAT32 і NTFS розмір кластера звичайно дорівнює 4 КБ.

Файл БД, розміщений на сервері, доступний усім клієнтам при запущеному сервері *InterBase*, причому навіть у тому випадку, якщо цей файл не є загальним мережним ресурсом. Тому рекомендується не робити файл БД доступним по мережі з метою захисту його від несанкціонованого доступу і випадкового пошкодження.

Цікавим є той факт, що на сервері *InterBase* крім файлу ibserver.exe завжди запущений додатковий контролюючий процес – IBGuard.exe. Він забезпечує автоматичний перезапуск сервера *InterBase* у випадку його аварійної зупинки.

14.3 Багатоверсійна архітектура *InterBase*

Багатоверсійна архітектура – це нова ідея в теорії проектування СУБД, уперше реалізована в *InterBase*. Завдяки багатоверсійній архітектурі вдалося ефективно реалізувати режим роботи, при якому читаючі користувачі не блокують пишучих. Крім того, з'явилася можливість швидко відновлювати БД після збоїв, а також домогтися багатьох інших переваг.

Сутність багатоверсійної архітектури проста і полягає в наступному.

Усі зміни даних у записах виробляються не в самих записах, а в їхніх копіях. Як тільки користувач підключився до БД і запустив транзакцію, у якій робляться якісь зміни в даних, для цього користувача створюється своя версія записів, у яких повинні робитися зміни. Ця версія стає власністю даної транзакції, і всі операції в ній будуть вироблятися з цією версією.

Якщо транзакція підтвердилася, то колишня (вихідна) версія запису буде позначена як видалена. Нова версія запису позначається як основна. Саме цю версію запису побачать інші транзакції, що будуть запущені пізніше.

Якщо відбудеться відкат транзакції, то нова версія буде відзначена як видалена, і основною версією залишиться колишня версія запису. Для всіх інших користувачів нічого не зміниться.

Тепер припустимо, що після запуску першої транзакції до її завершення запускається друга транзакція, що намагається прочитати той же запис, що змінює перша транзакція. Друга транзакція буде бачити попередню версію запису.

Отже, ідея багатоверсійності дуже проста – кожній транзакції дати свою версію запису. Читаючий користувач не заважає пишучому користувачу.

Але якщо дві транзакції будуть намагатися змінити той самий запис, то виникає конфлікт, розв'язування якого виробляється з використанням механізму блокувань, суть яких була розглянута вище.

При багатоверсійній архітектурі постійно накопичуються застарілі версії записів, що називають "сміттям". Очевидно, що ці версії записів підлягають видаленню. Процес видалення застарілих версій записів називають *збиранням сміття*.

Збирання сміття виробляється щораз, коли яка-небудь транзакція побажає прочитати даний запис. Ця транзакція зчитує всі існуючі версії даного запису і застарілі версії, що не використовуються в даний момент іншими транзакціями, видаляє.

Таким чином, видно, що процес збирання сміття є *кооперативним*, тобто в ньому можуть брати участь у різні моменти часу всі транзакції.

14.4 Типи даних у InterBase

У *InterBase* існують 12 типів даних, що задовольняють практично будь-яким вимогам розроблювачів БД. Ці типи можна згрупувати в такі 6 груп типів:

- для збереження цілих чисел: INTEGER, SMALLINT;
- для збереження дійсних чисел: FLOAT, DOUBLE PRECISION;
- для чисел з фіксованим записом: NUMERIC, DECIMAL;
- для збереження дати, часу, дати/часу: DATE, TIME, TIMESTAMP;
- для збереження символів: CHARACTER (скорочено CHAR), VARYING CHARACTER (скорочено VARCHAR);
- для збереження довільних динамічно розширюваних даних: BLOB.

Приведемо тепер характеристики цих типів даних більш докладно.

SMALLINT (2 байти) – цілі числа в діапазоні від –32768 до +32767;

INTEGER (4 байти) – цілі числа в діапазоні від –2 147 483 648 до + 2 147 483 647;

FLOAT (4 байти) – дійсні числа до 7 значущих цифр у діапазоні від 3.4·10-38 до 3.4·10+38;

DOUBLE PRECISION (8 байт) – дійсні числа до 15 значущих цифр у діапазоні від 1.7·10-308 до 1.7·10+308;

NUMERIC чи DECIMAL (перем.) – дійсні числа з фіксованою крапкою. Кількість значущих цифр і кількість цифр після крапки вказується при визначенні типу даних;

CHAR[ACTER](n) (0-32767 байт) – текстовий стовпець довжиною до n символів;

CHAR[ACTER](n) VARYING (0-32767 байт) – текстовий стовпець перемінної довжини, що містить до n символів;

DATE (8 байт) – дата в межах від 01.01.0100 до 11.12.5941. Можуть також зберігатися дані про час;

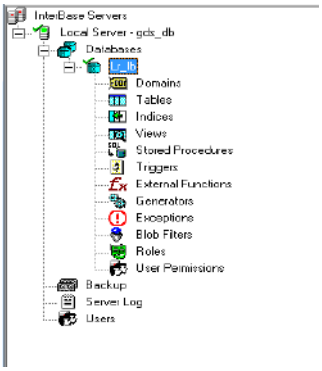
BLOB (перем.) – будь-які двійкові дані, що не можуть зберігатися в полях зазначених вище типів – наприклад, малюнки, музичні файли й ін.

Тип даних стовпців визначається в *SQL*-операторах при створенні таблиць (CREATE TABLE), при створенні доменів (CREATE DOMAIN), і при зміні структури таблиць (ALTER TABLE).

У *InterBase* не визначені типи даних "логічний" і "автоінкрементний" (лічильник). Логічний тип заміняється типом CHAR(1), а замість автоінкрементного типу спільно використовуються генератори і тригери, що забезпечують одержання унікальних значень при введенні даних. Кожне автоінкрементне поле зв'язане з генератором і тригером. При вставці запису тригер перевіряє значення цього поля, і якщо воно виявиться порожнім, вставляє в нього значення генератора, при цьому значення генератора збільшується на одиницю.

14.5 Склад БД *InterBase*

На відміну від локальних БД, побудова БД на *SQL*-сервері, у тому числі і на сервері *InterBase*, більш складна. БД на *SQL*-сервері містить у собі такі елементи:



- таблиці (Tables);
- індекси (Indexes);
- обмеження (Constraints);
- домени (Domains);
- перегляди (Views);
- генератори (Generators);
- тригери (Triggers);
- збережені процедури (Stored Procedures);
- привілеї (Roles)і ін. елементи.

Коротко охарактеризуємо призначення цих елементів.

Таблиці – це, як і раніше, основні елементи структури БД, призначені для збереження даних. Зв'язки між таблицями, засновані на використанні первинних і зовнішніх ключів, забезпечують високу безпеку і цілісність даних.

Індекси – як і в локальних БД, це додаткові структури даних, що забезпечують високу швидкість обробки даних.

Обмеження – це спеціальні засоби, що дозволяють автоматизувати процес уведення даних, запобігати помилок введення, керувати порядком сортування записів.

Домен – це іменований опис стовпця. Його можна розглядати як свого роду шаблон опису стовпця. Один раз визначивши домен, його ім'я можна потім можна багаторазове використовувати для створення інших стовпців (у тому числі і для різних таблиць).

Перегляд (чи **Представлення**) – це віртуальна таблиця, записи в який відібрані за допомогою оператора SELECT. Перевага перегляду полягає в тому, що створивши його, у наступному можна використовувати його безпосередньо без повторних звертань до оператора SELECT.

Генератор – це засіб для одержання унікальних цілих значень. У *InterBase* відсутній автоінкрементний тип даних, призначений для одержання унікальних значень у ключових полях. Для цієї мети використовуються генератори.

Збережена процедура – це підпрограма, розташована на сервері і викликувана з програми клієнта. Використання збережених процедур збільшує швидкість обробки даних. Їхня перевага також полягає в тому, що вони є загальними для всіх клієнтських додатків, завдяки чому різко скорочується число помилок, внесених у дані.

Тригер – це збережена процедура, що викликається автоматично при модифікації записів у БД, тобто при зміні, видаленні і додаванні записів. Тригери від звичайних збережених процедур відрізняються тим, що їх не можна викликати з додатка клієнта, передавати їм параметри й одержувати від них результати.

Привілегії – це спеціальні структури даних, у яких зберігається інформація про права доступу для кожного з зареєстрованих користувачів. Після створення об'єкта (наприклад, таблиці) доступ до нього має тільки його творець. Для надання можливості доступу до даних інших користувачів для них потрібно призначити відповідні привілеї. Передбачено такі рівні привілеїв:

ALL – усі права доступу;
SELECT – тільки читання;
DELETE – видалення;
INSERT – вставка;
UPDATE – модифікація.

14.6. Програма IBConsole

14.6.1. Призначення програми

Програма *IBConsole* призначена для керування сервером *InterBase*. Виклик програми можна виконати через головне меню *Windows*: **Пуск | Програми | InterBase | IBConsole** (чи через ярлик, якщо він створений).

Програма *IBConsole* забезпечує:

- Керування локальним і віддаленим сервером;
- Керування БД;
- Інтерактивне виконання *SQL*-запитів.

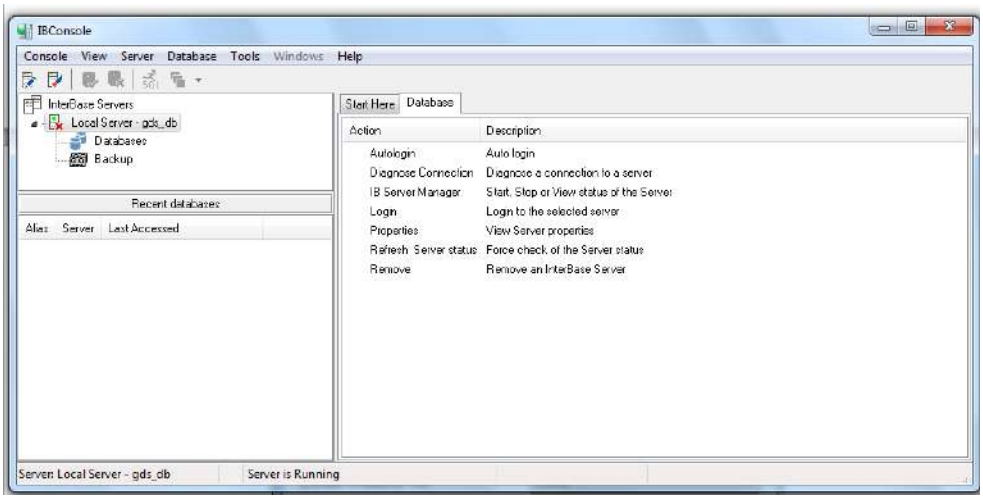


Рисунок 14.1- Вікно програми *IBConsole*

У вікні розміщені дві панелі. У лівій панелі відображаються у виді дерева всі зареєстровані сервери і їх БД. У правій панелі можуть відображатися інформація, що відноситься до елемента, обраного у лівій панелі, чи перелік команд (дій), що можуть бути виконані у поточному стані.

14.6.2 Керування сервером

Керування сервером включає такі дії:

- реєстрація сервера;
- підключення сервера;
- перегляд протоколу роботи;
- керування сертифікатами;
- визначення користувачів.

При запуску програми відкривається вікно, вид якого показаний на рис. 14.2.

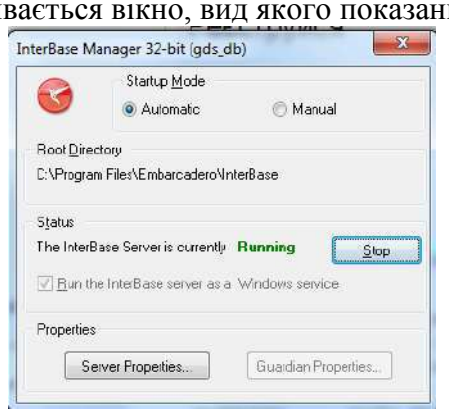


Рисунок 14.2 - Вікно InterBase Manager

Підключення до сервера виробляється в такий спосіб. Виділити значок потрібного сервера і виконати команду Login…(за допомогою контекстного меню чи подвійним щигликом на значку сервера). Відкриється вікно Server Login, вид якого показаний на рис. 14.3. У цьому вікні потрібно ввести ім'я користувача (User Name) і його пароль (Password).

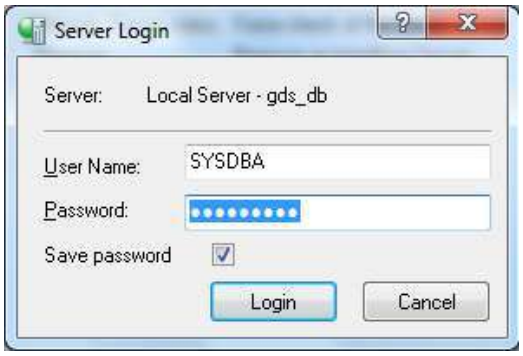


Рис. 14.3 - Вікно з'єднання із сервером

Уведіть з клавіатури SYSDBA і masterkey – ім'я і пароль адміністратора БД, і після цього натисніть кнопку **Login**. Після встановлення з'єднання біля значка сервера з'явиться зелена галочка. Під значком сервера розгорнеться гілка дерева, що містить значки об'єктів, що маються на сервері (Рис. 14.4).

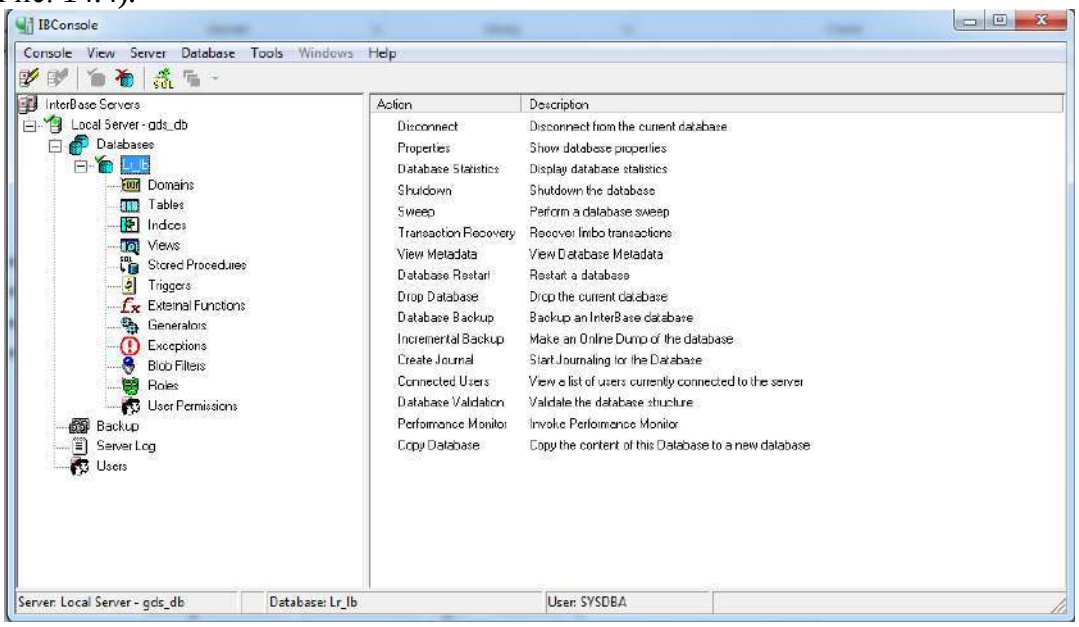


Рисунок 14.4 - Установлено з'єднання з локальним сервером

Для розриву з'єднання із сервером необхідно виконати команду **Server | Logout**.

14.6.3 Керування базами даних

Керування БД включає такі дії:
- реєстрація БД;
- підключення до БД;
- створення і видалення БД;
- перегляд метаданих;
- збір сміття;
- перевірка стану БД;
- аналіз статистики;
- збереження і відновлення БД.

Розглянемо деякі з цих операцій.

Для створення БД необхідно виконати команду меню **Database | Create Database…** . Відкриється вікно створення БД, вид якого приведений на рис. 14.5.

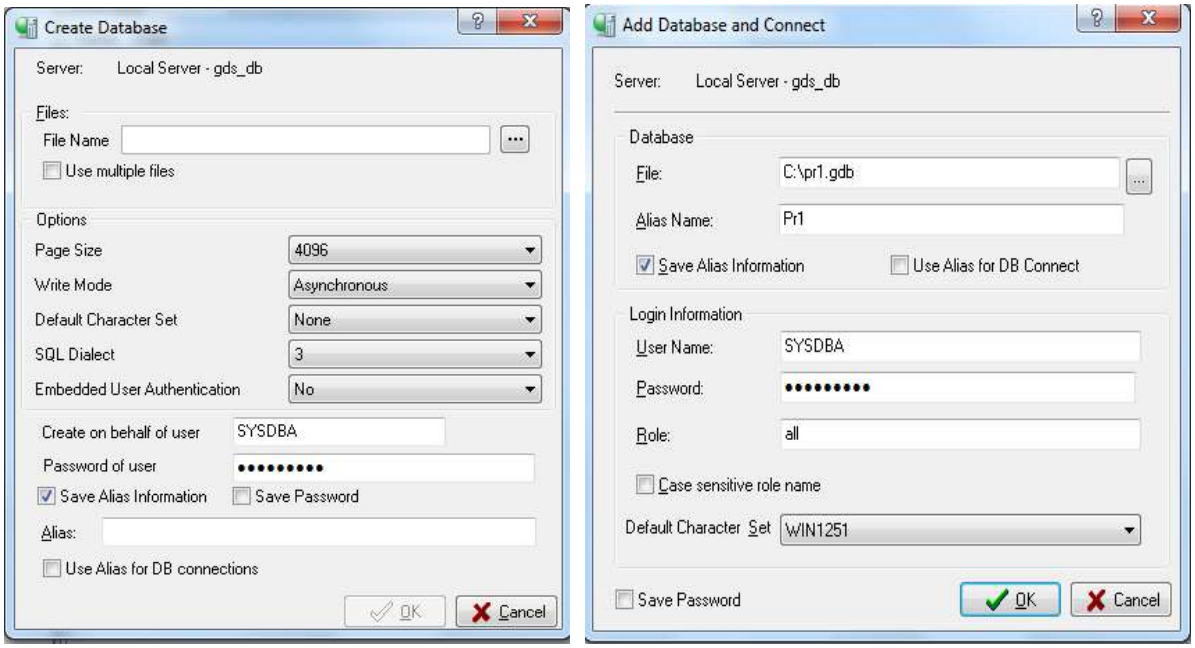


Рисунок. 14.5 - Створення нової БД

У цьому вікні необхідно ввести:

- у стовпці File(s) увести повне ім'я файлу створюваної БД;
- у поле Alias увести псевдонім БД, призначений для її ідентифікації усередині сервера (не зв'язаний із псевдонімом **BDE**).

Серед параметрів (Options) становить інтерес установка набору для кодування символів (Default Character Set). Якщо передбачається використовувати символи кирилиці, то необхідно установити набір WIN1251.

Після введення цих даних натиснути кнопку **OK**. З тільки що створеною БД відразу встановлюється з'єднання. Під значком створеної БД розгорнеться гілка зі значками об'єктів (поки порожніх), що автоматично включаються в БД. Вид вікна *IBConsole* буде приблизно такий, як це показано на рис. 14.5.

Для підключення до уже наявної БД необхідно виділити значок потрібної БД і потім виконати команду **Database | Connect**. Для відключення від БД виконати команду **Database | Disconnect**.

14.6.4 Інтерактивне виконання SQL-запитів

Програма *IBConsole* дозволяє виконувати запити мовою *SQL* в інтерактивному режимі. Для переходу в цей режим необхідно виконати команду меню **Tools | Interactive SQL…** чи натиснути однойменну кнопку  на панелі інструментів. Вид екрана в режимі Interactive SQL показаний на рис. 14.6.

Вікно складається з чотирех частин. Верхня частина призначена для введення *SQL*-запитів. У вікні можна ввести і відразу виконати різні *SQL*-оператори, включаючи створення і видалення БД, таблиць, переглядів і ін.

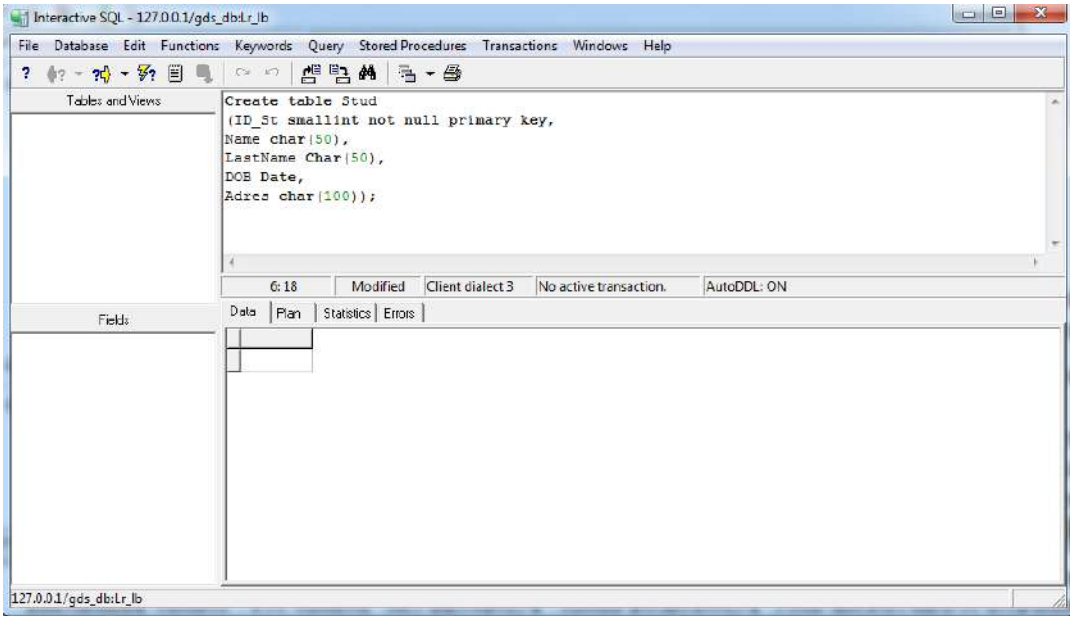
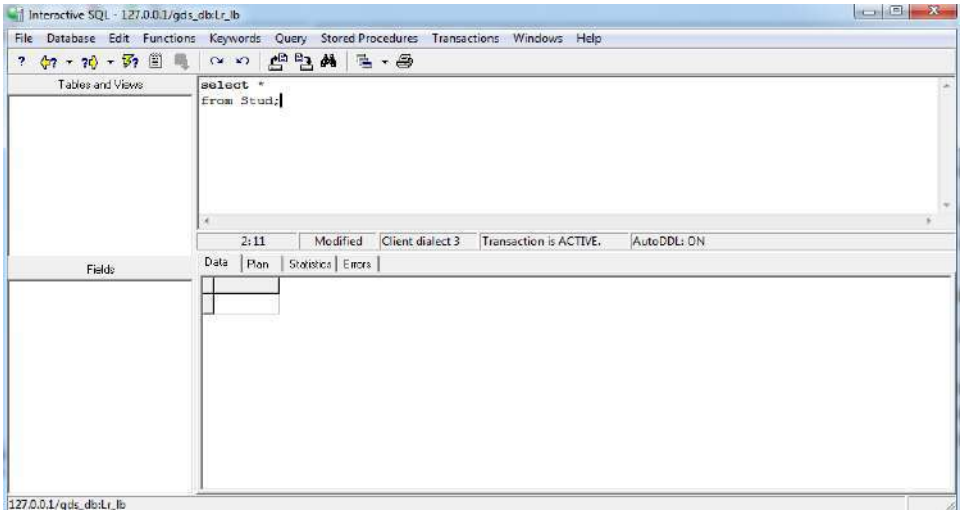


Рисунок. 14.6 - Вікно Interactive SQL

У вікні на рис. 14.6 введений оператор Create для створення таьлици STUD.

Для запуску запиту на виконання потрібно натиснути кнопку (чи виконати команду меню **Query | Execute**).

Якщо виконуэться оператор SELECT для перегляду змісту таблиці STUD то у нижній частині вікна відображаються результати виконання запиту. Усі запити, що вводяться, запам'ятовуються. При необхідності кожний з запитів, що раніше виконувалися, можна відкрити і, якщо потрібно, відкоригувати.



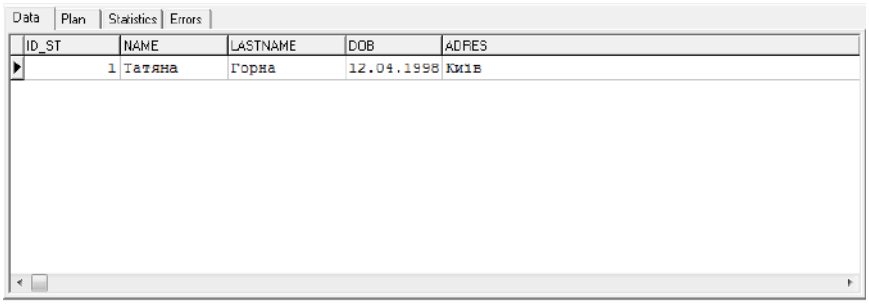


Рисунок 14.7 - Вид екрана в режимі виконання *SQL*-запитів

Для вибору потрібного запиту можна скористатися кнопками і , розташованими на панелі інструментів. Результати виконання запитів можна зберігати в окремому файлі, викона-

вши команду **Query**│**Save Output**. Можна також зберегти у файлі всю історію роботи з БД, виконавши команду **File**│**Save SQL History**. Потім можна відкрити його і при необхідності повторно виконати.

Контрольні питання:

1. Призначення й основні особливості архітектури “клієнт-сервер”.
2. Поняття транзакції.
3. Суть основних проблем, що виникають при роботі з БД багатьох користувачів.
4. Суть конфліктів при відновленні даних і їхнє вирішення за допомогою блокувань.
5. Рівні ізоляції транзакцій: їхнє призначення і характеристика.
6. У чому ідея і переваги багатоверсійної архітектури сервера *InterBase*.
7. Типи даних *InterBase* і їхня характеристика.
8. Основні об'єкти, що входять до складу БД *InterBase*, і їхнє призначення.
9. Призначення і можливості програми *IBConsole*.

15. Створення об'єктів БД і керування ними на сервері *InterBase*

15.1. Створення таблиць БД

15.1.1 Загальний формат оператора створення таблиць БД

Таблиця БД на сервері створюється за допомогою *SQL*-оператора CREATE TABLE. Формат оператора наступний:

CREATE TABLE ⟨ім'я таблиці⟩
((ім'я стовпця ((опис стовпця(((обмеження стовпця(,(
. . . . . .
((обмеження таблиці (,(
((опис ключа (,(
. . . . . .
((опис індексу (, ...(
);

З приведеного загального формату оператора видно, що створювана структура таблиці визначається такими елементами:

- опису стовпців;
- обмеження стовпців;
- опис ключів;
- опис індексів;
- обмеження таблиці.

Обов'язково повинні бути задані ім'я таблиці і, як мінімум, опис одного стовпця. Далі розглянемо детально деякі з цих елементів.

15.1.2. Опис стовпця

Опис стовпця має наступний формат:

⟨опис стовпця⟩ = ⟨ім'я стовпця⟩ {⟨тип даних⟩ |
COMPUTED [BY] (⟨вираження⟩) |
⟨домен⟩}

Відповідно до даного формату стовпець можна визначити одним з наступних трьох способів:

- задати ім'я і тип даних стовпця, наприклад:

KodStud SMALLINT,
Fam VARCHAR(15) ;

- створити стовпець, що обчислюється, наприклад:

Vozrast COMPUTED BY (2003-**GodRog)**

- створити стовпець на основі домену, наприклад:

Fam D_Fam ,

де **D_Fam** - ім'я домену, що створений раніше.

15.1.3. Обмеження стовпця

Обмеження стовпця є необов'язковим елементом, призначеним для автоматизації введення даних і запобігання помилок при введенні. Обмеження стовпця має наступний формат:

[DEFAULT {⟨значення⟩ | NULL | USER}]
[NOT NULL]
[COLLATE ⟨порядок сортування⟩]

[CHECK ⟨умова⟩]

Інструкція DEFAULT призначена для завдання значення за замовчуванням, що буде автоматично вводитися в стовпець щораз при додаванні в таблицю нового запису. Значення стовпця, що вводиться, можна задавати, можна задавати так:

- вводити константу (літерал);
- значенням NULL – у стовпець буде заноситися порожнє значення;
- значенням USER – у стовпець буде заноситися ім'я поточного користувача.

Інструкція NOT NULL указує, що стовпець не може містити порожнього значення, тобто в ньому обов'язково повинно бути введене значення припустимого типу і діапазону.

Інструкція COLLATE визначає порядок сортування для строкових типів даних.

В інструкції CHECK задаються умови на припустимі значення даних у стовпці. Формат завдання умов приблизно такої ж, як в інструкції WHERE оператора SELECT, що розглядався нами раніше.

15.1.4. Визначення ключів

У таблицях *InterBase* ключі підрозділяються на *первинні* й *унікальні* (для головних таблиць) і *зовнішні* (для підлеглих таблиць). Нагадаємо, що первинні ключі призначені для забезпечення унікальності записів у таблицях, а зовнішні – для зв'язування таблиць, завдяки чому забезпечується посилальна цілісність даних.

Для створення первинного ключа використовується інструкція:

PRIMARY KEY (⟨список стовпців ключа⟩) .

Ця інструкція вказується в операторі CREATE TABLE після описів усіх полів чи може безпосередньо вказуватися в описі окремих стовпців, що включаються до первинного ключа.

Приклади:

```
CREATE TABLE Predmet (
KodPred SMALLINT NOT NULL PRIMARY KEY,
NamPred CHAR(15)  );
CREATE TABLE Predmet (
KodPred SMALLINT NOT NULL ,
NamPred CHAR(15),
PRIMARY KEY(KodPred)  );
```

Обоє з приведених прикладів рівноцінні по результаті – створюється таблиця Predmet , у якій первинним ключем визначений стовпець **KodPred .**

Таблиця може мати тільки один первинний ключ, однак, крім нього, ще можна визначити кілька унікальних ключів. Для створення унікального ключа використовується інструкція UNIQUE наступного формату:

UNIQUE (⟨список стовпців ключа⟩) .

15.1.5. Визначення обмежень посилальної цілісності (створення зв'язків)

Нагадаємо, що дія обмежень посилальної цілісності укладається в тім, що вводиться заборона на виконання якихось з наступних дій:

з головної таблиці не можна видалити запис, якщо в підлеглій таблиці є зв'язані з ним записи;

у підлеглу таблицю не можна додати запис, не зв'язаний з жодним з записів головної таблиці;

не можна змінити значення в стовпцях, яки належать ключу зв'язку.

Обмеження посилальної цілісності задається в наступному форматі:

[CONSTRAINT ⟨ім'я обмеження⟩]

FOREIGN KEY (⟨список стовпців ключа⟩)

REFERENCES ⟨ім'я головної таблиці⟩
[⟨список стовпців ключа головної таблиці⟩]

Завдання обмеження посилальної цілісності полягає в тому, що ключу (первинному чи унікальному) головної таблиці ставиться у відповідність зовнішній ключ підлеглої таблиці.

В інструкції CONSTRAINT задається ім'я обмеження. Якщо ім'я не задається, воно формується сервером за замовчуванням.

Інструкція FOREIGN KEY указує стовпці підлеглої таблиці, що повинні бути її зовнішнім ключем. Очевидно, що кількість і тип стовпців, що утворять зовнішній ключ підлеглої таблиці, повинні відповідати кількості і типу стовпців первинного (чи унікального) ключа головної таблиці.

В інструкції REFERENCES указується головна таблиця

Приклад завдання обмеження:

CREATE TABLE Usp (
KodStud SMALLINT NOT NULL,
KodPred SMALLINT NOT NULL,
Ocenka SMALLINT NOT NULL,
PRIMARY KEY (**KodStud, KodPred**)
FOREIGN KEY (**KodPred**) REFERENCES Predmet);

Видалити обмеження посилальної цілісності можна наступним оператором:

ALTER TABLE ⟨ім'я таблиці⟩
DROP ⟨ім'я обмеження посилальної цілісності⟩

15.2. Створення індексів

Нагадаємо, що індекс призначений для забезпечення швидкого пошуку інформації в БД. Досягається це за рахунок використання індексно-послідовного методу доступу до даних.

Індекс створюється за допомогою оператора CREATE INDEX, що має наступний формат:

CREATE [UNIQUE] [⟨порядок сортування⟩] INDEX
⟨ім'я індексу⟩ ON ⟨ім'я таблиці⟩ (⟨ім'я стовпця⟩, …[⟨ім'я стовпця⟩]) ;

Описувач UNIQUE означає, що індекс вимагає унікальності значень стовпця, по якому він будується.

Описувач ⟨порядок сортування⟩ задається одним з наступних значень:

ASCENDING – сортування по зростанню (за замовчуванням);
DESCENDING – сортування по убуванню.

Приклад, у якому для таблиці Usp створюється індекс по полю KodPred, наступний:

CREATE INDEX
Ind_KodPred ON Usp(**KodPred**);

Видалити індекс можна за допомогою оператора:

DROP INDEX ⟨ім'я індексу⟩;

15.3. Створення генераторів і тригерів

15.3.1. Створення генератора

Генератор – це збережена процедура, призначена для одержання унікальних значень полів. Створення генератора включає два етапи. Спочатку створюється власне генератор за допомогою оператора:

CREATE GENERATOR ⟨ім'я генератора⟩

Потім установлюється його початкове значення оператором:

SET GENERATOR ⟨ім'я генератора⟩ TO ⟨початкове значення⟩

Виклик генератора виконується за допомогою функції:

GEN_ID(⟨ім'я генератора⟩, ⟨крок⟩);

Функція повертає значення, збільшене в порівнянні з попереднім значенням генератора на величину кроку. Величина кроку, як і значення генератора, може приймати тільки цілі значення .

Після створення генератора змінити його початкове значення чи величину кроку неможливо. Якби це допускалося, то це приводило б до появи повторюваних значень.

Приклад. Створимо генератор з ім'ям Gen_KodStud для одержання унікальних значень стовпця **KodStud** у таблиці Student**:**

CREATE GENERATOR Gen_KodStud ;

SET GENERATOR Gen_KodStud TO 1;

Генератор створений. Тепер його можна викликати за допомогою функції:

GEN_ID(Gen_KodStud , 1) ;

Виклик цієї функції провадиться в тригері, що розглядається нижче.

15.3.2. Створення тригера

Тригер являє собою процедуру, що знаходиться на сервері БД і викликається автоматично при модифікації записів, тобто при зміні значень стовпців, при видаленні чи додаванні записів. Створюється тригер за допомогою оператора CREATE TRIGGER, що має наступний формат:

CREATE TRIGGER ⟨ім'я тригера⟩ FOR ⟨ім'я таблиці⟩
[ACTIVE | INACTIVE]
[BEFORE | AFTER]
[UPDATE | INSERT | DELETE]
[POSITION ⟨число⟩]
AS ⟨тіло тригера⟩

Описувачі ACTIVE і INACTIVE визначають активність тригера відразу після його створення. За замовчуванням діє ACTIVE.

Описувачі BEFORE і AFTER задають момент початку виконання тригера – до чи після настання відповідної події, зв'язаної зі зміною у запису.

Описувачі UPDATE, INSERT і DELETE визначають, при настанні якої події викликається тригер – при редагуванні, додаванні чи видаленні запису.

Для однієї події можна створити кілька тригерів, послідовність виклику яких визначає число, зазначене в описувачі POSITION. Тригери виконуються в порядку зростання цих чисел.

Тіло тригера програмується так само, як і звичайна збережена процедура. Для цього використовується спеціальна мова, яку ми не будемо розглядати.

Приведемо наступний приклад:

```
CREATE TRIGGER BEF_INS_Stud FOR Student
ACTIVE  BEFORE  INSERT
AS
Begin
NEW.KodStud = GEN_ID(Gen_KodStud, 1);
End;
```

У цьому прикладі тіло тригера складається з одного оператора, виконання якого приводить до введення нового значення в поле **KodStud** таблиці Student. У лівій частині оператора розміщено предикат NEW, що вказує на нове значення полю **KodStud**. У правій частині записано виклик генератора, що створює унікальні значення для полю **KodStud.**

15.4. Маніпулювання даними

Під маніпулюванням даних розуміється відбір, редагування, уведення нових чи видалення записів.

Для відбору записів використовується оператор SELECT, що був вивчений нами раніше. Усі розглянуті раніше правила запису оператора SELECT для локальних БД можуть бути застосовні і для сервера БД.

Для редагування, додавання нових і видалення записів призначені оператори UPDATE, INSERT і DELETE.

Оператор редагування даних UPDATE має формат:

UPDATE ⟨ім'я таблиці⟩

SET ⟨ім'я полю⟩ = ⟨вираження⟩,

. . . . . .

[WHERE ⟨умова відбору⟩];

Для всіх записів, що задовольняють умові відбору, змінюються значення полів. Елемент ⟨вираження⟩ визначає нове значення, що буде привласнено стовпцю ⟨вираження⟩.

Приклад:

UPDATE Student

SET **GodRog** = 1980

WHERE **Fam =** "Іванов"

Оператор додавання записів INSERT має наступний формат:

INSERT INTO ⟨ім'я таблиці⟩

[⟨список полів⟩]

VALUES (⟨список значень⟩);

У таблицю ⟨ім'я таблиці⟩ додається новий запис. У доданому запису заповнюються поля, зазначені в списку полів ⟨список полів⟩ . Значення полів беруться зі списку значень, розташованого після слова VALUES. Список полів і список значень повинні відповідати один одному по типі і числу елементів. Порядок розташування полів може відрізнятися від порядку розташування їх у таблиці.

Приклад:

INSERT INTO Student (KodStud, Fam, Nam1, Nam2)

VALUES (1, "Іванов", "Іван", "Іванович");

Список полів в операторі INSERT може бути відсутній. У цьому випадку необхідно вказати значення всіх полів таблиці.

Оператор видалення записів DELETE має наступний формат:

DELETE FROM ⟨ім'я таблиці⟩

[WHERE ⟨умова відбору⟩];

З таблиці з ім'ям ⟨ім'я таблиці⟩ видаляються всі записи, що задовольняють заданій умові відбору. Якщо умова відбору не задана, з таблиці видаляються всі записи.

15.5. Створення БД за допомогою IBExpert

15.5.1. Загальні відомості про CASE-технології

Термін CASE (Computer Aided Software Engineering) використовується в даний час у дуже широкому змісті. Первісне значення терміна CASE, обмежене питаннями автоматизації розробки тільки лише програмного забезпечення, у даний час набуло нового сенсу, що охоплює процес розробки складних ІС у цілому. Тепер під терміном "CASE-засоби" розуміються програмні засоби, що підтримують процеси створення і супроводу ІС, включаючи аналіз і формулювання вимог, проектування прикладного програмного забезпечення і баз даних, генерацію коду, тестування, документування, забезпечення якості, керування проектом, а також інші процеси. CASE-засоби разом із системним ПЗ і технічними засобами утворюють повне середовище розробки ІС.

CASE-технологія являє собою методологію проектування ІС, а також набір інструментальних засобів, що дозволяють у наочній формі моделювати предметну область, аналізувати цю модель на всіх етапах розробки і супроводу ІС і розробляти додатки відповідно до інформацій-

них потреб користувачів. Більшість існуючих CASE-засобів засновано на методологіях структурного чи об'єктно-орієнтованого аналізу і проектування, що використовують специфікації у виді діаграм чи текстів для опису зовнішніх вимог, зв'язків між моделями системи й архітектури програмних засобів.

Найбільш поширеним CASE-засобом для проектування БД є програма ERWIN. Програма Microsoft Visio 2002 також містить CASE-засоби для проектування БД.

Але у цих програм є один спільний недолік – вони не повністю сумісні з системою Interbase. Тому ми будемо розглядати CASE-засіб Database Designer, який входить до складу програми IBExpert.

15.5.2. Пряме та обернене проектування

Процес генерації фізичної структури бази даних з логічної моделі даних називається прямим проектуванням (Forward Engineering).

Процес генерації логічної моделі з фізичної бази даних називається оберненим проектуванням (Reverse Engineering). Програма IBExpert дозволяє швидко створити модель даних шляхом оберненого проектування наявної бази даних.

IBExpert може зробити обернене проектування існуючої бази даних, автоматично створюючи відповідну діаграму моделі даних. Після того як діаграма буде згенеровано, ви можете, використовуючи інструменти IBExpert, додавати нові об'єкти і перепроектувати структуру бази даних.

Пряме проектування виконують для того, щоб з логічної моделі даних перейти до інструкцій мови SQL, за допомогою яких можна внести зміни в БД.

Обернене проектування виконують для:

- кращого розуміння структури БД;
- створення документації;
- подальшого внесення змін в структуру БД шляхом прямого проектування.

15.5.3. Використання CASE-засобу Database Designer програми IBExpert

У складі програми IBExpert міститься CASE-засіб Database Designer. Він дозволяє розробити модель даних - структурну схему БД. Також він може виконувати пряме і обернене проектування БД.

Для запуску Database Designer треба виконати команду Инструменты | Database Designer з меню програми IBExpert.

Для виконання оберненого проектування БД потрібно виконати команду Designer | Reverse Engineer. На екрані з'явиться список зареєстрованих в програмі IBExpert баз даних. Виберіть одну з них. У наступному вікні виберіть режим оновлення діаграми - Create new diagram (створити нову діаграму) чи Update current diagram (оновити поточну діаграму), після чого натисніть кнопку Start. На діаграмі з'явиться схема БД, представлену на рис.15.1.

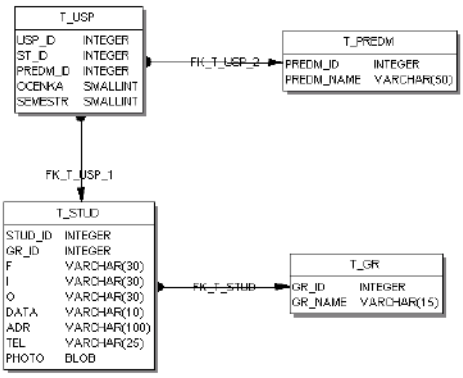


Рисунок 15.1 - Схема бази даних успішності студентів, створена під час оберненого проектування.

Для виконання прямого проектування БД треба додати на діаграму необхідні об’єкти БД і відредагувати їх властивості, створивши структурну схему. Як це робиться, ви дізнаєтесь на лабораторних заняттях. Структурна схема БД з двох таблиць може виглядати так, як показано на рис. 15. 2.

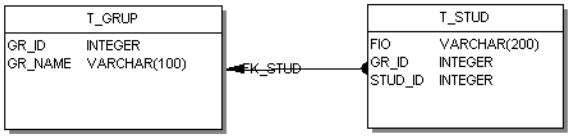


Рисунок 15.2 - Структурна схема БД з двох таблиць.

Після цього потрібно виконати команду Designer | Generate Script і вказати файл для збереження SQL-скрипта. Після закінчення генерації SQL-скрипта вам буде запропоновано завантажити файл скрипта у SQL-редактор. Якщо ви зробите це, а потім натиснете клавішу F9 (виконати скрипт), зміни будуть внесені в БД.

15.6. Система автоматизованого програмування БД IBExpert

Для автоматизації проектування БД та розробки інформаційних систем використовуються програмно-технологічні засобі які мають назву CASE(Computer Aided Software Engineering). Програма IBConsole, яка входить до складу Interbase, має досить обмежені можливості. На думку багатьох адміністраторів БД, IBConsole є не дуже зручним інструментом адміністрування. Особливо чітко її недоліки проявляються при створенні та адмініструванні складних баз даних. Тому найчастіше для роботи з базами Interbase використовують продукти сторонніх розроблювачів, що забезпечують зручний графічний інтерфейс і додаткові засоби проектування та адміністрування БД. Одним з таких програмних продуктів є програма-оболонка IBExpert.

Програма-оболонка IBExpert призначена для створення, проектування й адміністрування БД, а також для редагування даних, що містяться в ній. До складу оболонки входять також SQL-редактор, візуальні засоби побудови SQL-запитів і засоби адміністрування. Нові версії програми IBExpert мають також CASE-засоби проектування БД.

IBExpert дає можливість виконувати складні операції проектування БД і при цьому не потребує знання мови SQL. Адміністратор БД виконує більшість операцій на рівні візуального об'єктно-орієнтованого проектування, а програма IBExpert перетворює ці дії в інструкції мовою SQL, які виконуються сервером Interbase.

15.6.1. Склад CASE- засобу IBExpert

До складу програми IBExpert входять наступні основні компоненти.

DB Explorer – це вікно, в якому відображені всі зареєстровані в програмі IBExpert бази даних, а для підключених БД – також список об’єктів кожної БД.

SQL-редактор призначений для введення, редагування та виконання інструкцій мовою SQL.

Query Builder призначений для візуальної побудови SQL-запитів.

SQL-монітор дозволяє спостерігати за всіма чи вибраними інструкціями SQL, які виконуються.

Менеджер користувачів дозволяє додавати і видаляти користувачів, зареєстрованих на сервері, а також змінювати їх паролі. Нагадаємо, що інформація про користувачів зберігається на сервері в базі даних ISC4.GDB.

Менеджер прав призначений для надання і редагування прав доступу окремих користувачів до об’єктів БД.

Менеджер файлів дозволяє розділити базу на кілька файлів. Звичайно це виконують тоді, коли розмір файлу БД перевищує 2 ГБ.

Database Designer – це CASE-засіб проектування БД.

Генератор тестових даних дозволяє заповнювати таблиці БД випадковими або невипадковими даними. Звичайно його використовують тоді, коли необхідно протестувати БД з великою кількістю даних, а реальних даних на цей момент ще немає (таблиці порожні).

Утиліти резервного копіювання та аварійного відновлення дозволяють виконувати резервне копіювання БД та її відновлення в разі пошкодження. Ці операції можна виконувати без зупинки сервера, тобто вони не заважають користувачам, що працюють з БД.

Утиліта перевірки БД виконує перевірку бази даних.

Утиліта статистики БД призначена для збору статистики БД.

Утиліта діагностики комунікацій дозволяє виявити і усунути помилки в роботі мережі, в якій працює віддалений сервер Interbase.

15.6.2. Створення та реєстрація БД за допомогою програми IBExpert

Для створення БД в програмі IBExpert необхідно виконати команду *База данных \ Создать базу*. З'явиться вікно, в якому необхідно вказати ім'я файлу бази даних і деякі додаткові параметри, такі як ім'я користувача, пароль, розмір сторінки та ін.. Після цього натисніть кнопку ОК.

Для роботи з базою даних її необхідно зареєструвати в оболонці IBExpert. При створенні нової бази програма IBExpert сама запропонує виконати реєстрацію БД. При реєстрації необхідно ввести повний шлях до файлу бази даних, опис, під яким вона буде відображена в оболонці, ім'я і пароль користувача, номер кодової сторінки і повний шлях до системної бази даних ISC4.GDB. Сервер Interbase використовує базу даних ISC4.GDB для збереження інформації про імена і паролі користувачів. Звичайно вона знаходиться в робочому каталозі сервера Interbase.

Щоб система Interbase правильно відображала кирилицю в записах бази даних, необхідно при створенні БД вказати кодову сторінку WI1251. Після реєстрації база даних буде відображена в списку зареєстрованих баз даних.

15.6.3. Редагування та адміністрування БД за допомогою програми IBExpert

Для роботи з базою даних необхідно підключитися до неї. Для цього двічі клацніть на назві бази даних у списку чи виберіть у меню "База данных" пункт "Подключиться к базе". Можна також натиснути відповідну кнопку на панелі інструментів.

Після підключення до бази даних під її назвою з'явиться список об'єктів, які може містити база даних. Спочатку БД не містить ніяких об’єктів.

Щоб створити таблицю, клацніть правою кнопкою миші на пункті "Таблицы" у списку об'єктів бази даних і виберіть у контекстному меню пункт "Новый объект". З'явиться вікно створення нової таблиці, в якому можна вказати назву таблиці та створити поля. Після створення структури таблиці необхідно внести зміни в базу даних. Для цього натисніть кнопку Compile на інструментальній панелі чи комбінацію клавіш Ctrl+F9. Програма IBExpert автоматично згенерує всі SQL-інструкції, що повинні бути виконані сервером Interbase для створення цієї таблиці. Вони будуть показані в окремому вікні. Натисніть кнопку Commit у цьому вікні для внесення змін у структуру бази даних.

Контрольні питання:

1. У чому ідея і переваги багатоверсійної архітектури сервера *InterBase*.
2. Типи даних *InterBase* і їхня характеристика.
3. Основні об'єкти, що входять до складу БД *InterBase*, і їхнє призначення.
4. Яку архітектуру має СКБД InterBase?
5. З яких частій складається СКБД InterBase?
6. Яке розширення мають файли баз даних InterBase?
7. Яки основні типи даних використовуються в InterBase?
8. Як налаштовуються зовнішні зв’язки ?

16. Сервер MySQL

16.1 Загальні відомості

MySql це вільна реляційна система управління базами даних, розробку та супровід якої виконує корпорація Oracle. Розробник програми MySQL - Майкл Віденіус (Michael Widenius засновник компанії MySQL AB (яку поглинала компанія SUN, а потім Oracle). Майкл Віденіус заснував нову компанію - Monty Program Ab. Соавтор -Петро Зайцев - експерт по продуктивності MySQL, колишній темлідер групи High Perfomance в MySQL, ведучий блог ySQLPerformanceBlog.com.

Сервер MySQL на даний момент найбільш поширений при організації динамічних сайтів або порталів в мережі Інтернет.

Причини популярності:

- Доступність і відкритий код програмного забезпечення.
- Система написана на C і C ++. Базова платформа Solaris 2.7-2.8, SuSE Linux 7.1 (ядро 2.4, ReiserFS), працює в AIX, BSDI, DEC Unix, FreeBSD, HP-UX, Linux 2.0, Mac OS X, SunOS, SCO OpenServer, Win7.
- Працює як багатопотокова система.
- Має прикладну інтерфейс практично для всіх систем та програмного забезпечення, що застосовуються для розробки Web-приложений: C, C ++, Java, Eiffel, Perl, PHP, Python.
- Має ODBC-драйвери.
- Організація швидкого доступу до даних, що особливо важливо для Інтернету.
- Таблиці організовані в вигляді B-дерева з тисненням індексу.
- До 32 індексів на таблицю, 16 колонок на індекс. Довжина індексу до 500 байт.
- Допускає дуже великі розміри таблиць.
- Підтримує 6 типів організації таблиць: ISAM, MyISAM, BerkeleyBD (BDB), MERG, HEAP.
- В версіях (починає з 4.1) з таблицями типу InnoDB працюють транзакції.
- Блокіровки виконуються на рівні строк

MySQL пропонує 6 типів ядра БД (движків) зберігання даних. Кожен движок зберігання даних повністю різноманітний і спроектований для конкретних потреб додатків.

MyISAM: Движок по замовчанню. Не підтримує транзакції, середня надійність зберігання даних. Чудова продуктивність читання даних для інтенсивних додатків. Більша частина веб-сервісів та зберігачів даних активно використовує MyISAM.

HEAP: Все в пам'яті. Дуже швидкий пошук даних, проте всі вони зберігаються тільки в пам'яті і будуть втрачені при зупинці сервера. Чудово підходить для тимчасових таблиць.

ARCHIVE: Використовується для зберігання великих обсягів даних без індексів, займає менший розмір.

Merge: Колекція MyISAM таблиць логічно об'єднані разом для єдиного представлення.

InnoDB: транзакційній тип движка, що застосовується при інтенсивних операціях запису, можливість блокування рівня рядків таблиці. Чудова відновлюваність і висока надійність зберігання даних. Призначено для "важких" приложений, що вимагають активного запису в конкуруючому середовищі

NDB: Кластерній движок - дані автоматично розділяються і репліцируються по різним машинам, що мають назву - дата вузлів. Використовується для додатків, які вимагають високої продуктивності з найвищою ступенем доступності. NDB добре працює на системах, що вимагають високої віддачі на операції читання.

Приклади використання двигунів зберігання для різних завдань:
- Пошуковий движок - NDB кластер
- Фінансові транзакції - InnoDB
- Сесійні дані - MyISAM або NDB кластер
- Локальні розрахунки - HEAP
- Словники – MyISAM.

Таблиця 16.1 Основні типи даних атрибутів таблиць.

Тип даних	Використання	Діапазони
TINYINT	Дуже маленьке ціле число	Діапазон числа зі знаком від -128 до 127. Діапазон числа без знака (unsigned) від 0 до 255.
SMALLINT	Маленьке ціле число	Діапазон числа зі знаком від –32768 до 32767. Діапазон числа без знака (unsigned) от 0 до 65535.
MEDIUMINT	Середнє ціле число	Діапазон числа зі знаком від –8388608 до 8388607. Діапазон числа без знака (unsigned) от 0 до 16777215.
INT или INTEGER	Ціле число	Діапазон числа зі знаком від –2147483648 до 2147483647. Діапазон числа без знака (unsigned) от 0 до 4294967295.
BIGINT	Велике ціле число	Діапазон числа зі знаком від –9223372036854775808 до 9223372036854775807. Діапазон числа без знака (unsigned) от 0 до 18446744073709551615.
FLOAT	Мале (одинарної точності) число з плаваючою комою. Не може бути числом без знака	Діапазони від –3.402823466E+38 до –1.175494351E-38, 0 и 1.175494351E-38 до 3.402823466E+38. Якщо кількість знаків після коми не встановлено або <= 24 це число з плаваючою комою одинарної точності.
DOUBLE, DOUBLE PRECISION, REAL	Нормальне (подвійний точності) число з плаваючою комою. Не може бути числом без знака	Діапазони від -1.7976931348623157E+308 до -2.2250738585072014E-308, 0 и 2.2250738585072014E-308 до 1.7976931348623157E+308. Якщо кількість знаків після коми невстановлених або 25 <= кількість знаків <= 53 означає чіслоа з плаваючою комою подвійної точності.
DECIMAL, NUMERIC	Розпаковане число з плаваючою комою від в 2- або 4 х цифровому вигляді (4 цифри по-замовчуванню)	Працює подібно типу даних CHAR: розпакований означає, що число зберігається у вигляді рядка, використовуючи один символ для кожної цифри - значення. Символ десяткової коми і символ негативного числа "-" не враховується в довжину. Якщо десяткове значення дорівнює 0, значення не матиме десяткової коми або дробової частини.

Тип даних	Використання	Діапазони
DATE	Дата	у вигляді «YYYY-MM-DD» (ГГГГ-ММ-ДД).
DATETIME	Дата і час	у вигляді «YYYY-MM-DD HH:MM:SS» (ГГГГ-ММ-ДД ЧЧ-ММ-СС).
TIMESTAMP	Дата і час	у вигляді «YYYYMMDDHHMMSS» (TIMESTAMP(14)), «YYMMDDHHMMSS» (TIMESTAMP(12)), «YYYYMMDD» (TIMESTAMP(8))
TIME	Час	Діапазони від «-838:59:59» до «838:59:59» у вигляді «HH:MM:SS»
YEAR	Рік в 2- або 4 х цифровому вигляді (4 цифри по замовчуванню)	у вигляді «YYYY».
CHAR	Рядок фіксованої довжини, яка справа доповнюються пробілами до зазначеної довжини, при зберіганні	Діапазон довжини від 1 до 255 символів. Прогалини видаляються, коли значення витягується. Значення CHAR упорядковано і порівнюються без урахування регістра в залежності від кодування за замовчуванням, якщо не встановлено прапор BINARY.
VARCHAR	Рядок змінної довжини. Примітка: конечні прогалини видаляються при збереженні (на відміну від специфікації ANSI SQL).	Діапазон довжина від 1 до 255 символів. Значення VARCHAR упорядковано і порівнюються без урахування регістру, якщо не встановлено прапор BINARY.
TINYBLOB, TINYTEXT		BLOB або TEXT з максимальною довжиною 255 (2 ^ 8 - 1) символів.
BLOB, TEXT		BLOB або TEXT з максимальною довжиною 65535 (2^16 - 1) символів.
MEDIUMBLOB, MEDIUMTEXT		BLOB або TEXT з максимальною довжиною 16777215 (2^24 - 1) символів.
LONGBLOB, LONGTEXT		BLOB або TEXT з максимальною довжиною 4294967295 (2^32 - 1) символів.
ENUM	Перерахування	Рядок-об'єкт, який може приймати тільки одне значення, вибирається зі списку значень «значення 1», «значення 2» або NULL. ENUM максимум може мати 65535 різних значень.
SET	Набір	Рядок-об'єкт, який може приймати нуль і більш значний, кожна з яких має бути вибрано зі списку значень «значення 1», «значення 2», ... Поле SET може мати максимум 64 варіанти значень.

При описі стовпців припустимі наступні нестандартні атрибути: AUTO_INCREMENT, ZEROFILL і UNSIGNED.

Атрибут AUTO_INCREMENT задає властивість Автоінкрементний описуваного полю. Крок позитивний і завжди дорівнює одиниці.

Тому поле з характеристикою AUTO_INCREMENT поводиться як тип даних «лічильник» в СУБД MS Access.

Характеристика ZEROFILL змушує MySQL доповнювати число попередніми нулями.

Наприклад, якщо стовпець заданий як INT (5) ZEROFILL, то значення 4 перетвориться в 00004.

Характеристика UNSIGNED свідчить про те, що ви будете працювати тільки з позитивними, беззнаковими числами.

Ядро InnoDB була розроблена **Хейккі Туурі** (фін. Heikki Tuuri) з компанії **Innobase** - фінського виробника СУБД. Це ядро було розроблено спеціально для великих таблиць. Розробники заявляють, що InnoDB - найшвидший з усіх відомих движків для БД заснованих на дисках (множинні тести це підтверджують).

InnoDB - одна з обираних підсистем низького рівня в СУБД MySQL, входить в усі стандартні збірки для різних операційних систем. Основною відмінністю InnoDB від інших підсистем низького рівня MySQL є наявність механізму транзакцій і зовнішніх ключів.

Підтримка InnoDB з'явилася в MySQL версії 3.23 в середині 2001 року як експериментальна. У версії 4.0 InnoDB входив в стандартну поставку, а починаючи з версії 5.5 став основним сховищем за замовчуванням. Сама СУБД доступна на умовах відкритої ліцензії. Після поглинання Innobase в 2005 році InnoDB стала продуктом корпорації Oracle.

Загальні характеристики ядра InnoDB сервера БД

- макс. диск: 64Тб;
- повна підтримка транзакцій (4 рівня ізоляції);
- блокування запису (не таблиці);
- два види блокування (SHARED, EXCLUSIVE);
- повнотекстовий індекс: немає;
- безпечна для транзакцій;
- індекси кластерізуються для «типових запитів»;
- підтримка цілісності (зовнішні ключі);
- може використовуватися на ОС з обмеженим розміром файлу;
- безліч налаштувань для оптимізації;
- дозволяє використовувати Raw Disk для таблиць в обхід файлової системи;
- за замовчуванням включений AUTOCOMMIT (SET autocommit = 1);
- автоматично детектив дедлокі (deadlocks).

Особливості ядра типу InnoDB:

1. Всі таблиці зберігаються в єдиному табличному просторі, тому імена таблиць повинні бути унікальні.
2. Зберігання даних в єдиному табличному просторі дозволяє зняти обмеження на обсяг таблиць. Файл з таблицями може бути розбитий на кілька частин і розподілений по декількох дисках або навіть хостів.
3. Таблиці підтримують автоматичне відновлення після збою.
4. Підтримуються транзакції.
5. Цей тип таблиць в MySQL єдиний, який підтримує каскадне видалення і зовнішні ключі.
6. Виконується блокування на рівні окремих записів.
7. Є розширена підтримка кодувань.

На відміну від таблиць MyISAM, де для кожної таблиці створюється один файл даних, дані InnoDB в настройках за умовчанням зберігаються в великих спільно використовуваних файлах (змінити це можна за допомогою налаштувань опції innodb_file_per_table), що дозволяє використовувати посторінковий кеш сторінок бази даних.

Формат даних InnoDB забезпечує надійне зберігання даних за рахунок транзакційності і блокування даних на рівні рядка.

Починаючи з версії MySQL 5.6.4, в Innodb доступний повнотекстовий пошук.

Постійна зберігання даних в таблицях об'ємом до 1 Тбайт і навантаженням на сервер до 800 вставок / оновлень в секунду.

Для збільшення продуктивності InnoDB використовують наступні установки (my.cnf):

innodb_open_files = 500
innodb_file_per_table
innodb_buffer_pool_size = 250M
innodb_flush_log_at_trx_commit = 2
innodb_thread_concurrency =8
innodb_lock_wait_timeout = 500
interactive_timeout = 20
back_log = 75
table_cache = 300
thread_cache = 32
thread_concurrency = 8
wait_timeout = 30
connect_timeout = 10

Percona server - це збірка MySQL (від **Петра Зайцева**) з включеним за замовчуванням **XtraDB storage engine**.

Відрізняється від MySQL + InnoDB plugin кращою продуктивністю / масштабованість, особливо на сучасних багатоядерних серверах. Також покращена функціональність - більше будь-якої корисної для оптимізації статистики та ін.

Збирається в варіантах базуються на MySQL 5.0 / 5.1. Повністю сумісний з таблицями innodb, тобто можна переходити від innodb до xtradb і назад без проблем (якщо не використовувати деякі специфічні для xtradb функції, типу меншого розміру сторінки).

Сховище XtraDB засноване на коді InnoDB-plugin, повністю сумісний з ним, але відрізняється помітно вищою продуктивністю, завдяки інтеграції патчів від компаній Google і Percona.

У XtraDB поліпшений механізм роботи з пам'яттю, покращено роботу підсистеми вводу / виводу InnoDB, додана підтримка декількох потоків читання та запису, підтримка управління пропускною спроможністю, реалізація упреждающей вибіркою даних (read-ahead), адаптивна установка контрольних точок (adaptive checkpointing), розширені можливості по масштабування для великих проектів, система організації блокувань адаптована для роботи на системах з великим числом CPU, додані додаткові можливості для накопичення та аналізу статистик (рис.16.1).

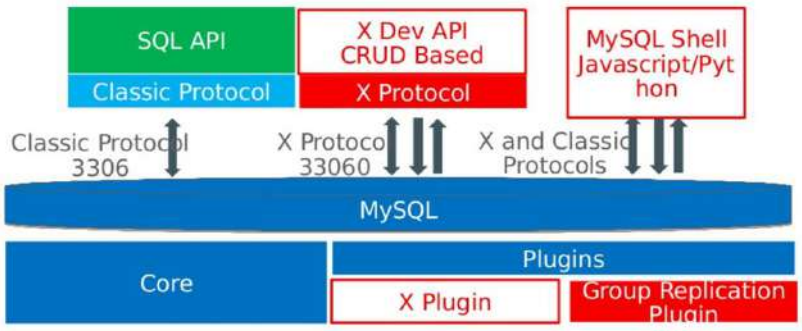


Рисунок 16.1 Архітекттура XTRADB

MariaDB - збірка від Monty Program Ab для платформ Windows, Debian, Ubuntu, RHEL 5, CentOS 5 и Solaris x86.

Синхронізована з кодовою базою MySQL і повністю з нею сумісний, може виступати в якості прозорої заміни MySQL 5.1, володіючи при цьому рядом розширених функцій, включаючи оптимізації продуктивності і з набором додаткових движків сховищ :

- Aria (Maria) - засноване на MyISAM високонадійне сховище, що відрізняються підвищеною стійкістю і збереженню цілісності даних після краху, при повній сумісності з MyISAM.

- OQGRAPH (сховище для організації складних графів).
- Sphinx - сховище для побудови пошукових движків.
- PrimeBase XT - опис російською, в якості заміни InnoDB використовується движок XtraDB.
- FederatedX - дозволяє організувати звернення до віддалених таблиць як до локальних.
- Патчі MyISAM - сегментований кеш (при високих навантаженнях дає істотний приріст).
- Віртуальні стовпці.
- Ліквідація таблиць - новий вид оптимізації запитів з використанням JOIN.
- Поліпшено механізми налагодження повільних запитів.

16.2 Створення і видалення БД

Для створення, видалення і редагування БД і їх об'єктів використовуються ***інструкції визначення даних***. Розглянемо їх роботу на прикладах. Для демонстрації будемо використовувати СУБД MySQL/MariaDB і утиліту phpMyAdmin на локальному сервері Apache.

Запит створення БД потрібно виконувати, «знаходячись» на сервері 127.0.0.1 (localhost), і відкривши консоль SQL (рис.16.2).

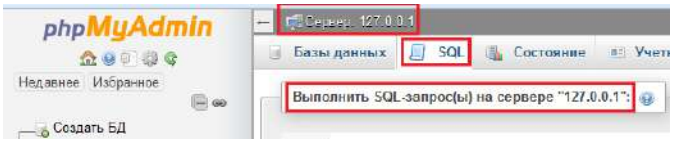


Рисунок 16.2 - Вікно консоль SQL

Localhost – за змістом означає «цей комп'ютер»; стандартне, офіційно зареєстроване доменне ім'я для приватних IP-адрес у діапазоні 127.0.0.1 – 127.255.255.255; для мережі, що складається тільки з одного комп'ютера, використовується лише одна IP-адреса 127.0.0.1.

Для виконання написаного в консолі запиту в утиліті phpMyAdmin, слід натиснути клавішу ***«Вперед»*** у вікні утиліти, або комбінацію клавіш клавіатури ***«Ctrl + Enter».***

Запит створення БД «database1» має вигляд:

```
CREATE DATABASE database1
```

Для коректного відображення інформації з БД, зокрема, кириличних символів**,** у нових версіях MySQL, при створенні БД слід задати кодування (charset) і collation (укр. «порівняння» або «представлення» сервера), наприклад для UTF-8:

```
CREATE DATABASE database1 DEFAULT CHARACTER SET utf8 COLLATE
                    utf8_general_ci
```

або для CP1251:

```
CREATE DATABASE database1 DEFAULT CHARACTER SET cp1251 COLLATE
cp1251_general_ci
```

Collation використовується, зокрема, при сортуванні.

Приклади *collations: cp1251_general_ci* – порівняння, не чутливе до реєстру символів; *cp1251_bin* – порівняння, чутливе до реєстру символів.

Кодування і порівняння можна задавати також при створенні (окремої) таблиці БД, або навіть поля. При цьому, прописується воно (аналогічно, як і для всього сервера; див. приклад) в кінці, після всієї іншої інформації відповідно про таблицю або поле. Раніше (до версії 4.1 MySQL) кодування задавалося для всього сервера одночасно, а тепер кодування можна задавати на кожному рівні ієрархії СУБД (сервер, БД, таблиці, поля).

Запит видалення БД «database1»:

DROP DATABASE database1

Для запуску виконання запиту натисніть кнопку **«Вперёд».**

16.3 Створення таблиць бази даних

Для створення таблиць використовується інструкція ***CREATE TABLE,*** в якій указуються наступні відомості:

– ім’я створюваної таблиці;
– імена стовпчиків, їхній тип даних, допустимість чи недопустимість значення NULL у стовпчиках тощо.

Ім’я таблиці повинне бути оригінальним, тобто в СУБД не повинно існувати однойменної таблиці, інакше буде виведене повідомлення про помилку. Якщо потрібно ***«перезаписати»*** одну таблицю іншою, то спочатку слід видалити вже існуючу таблицю, а потім створити нову.

Запит створення таблиці повинен обов’язково містити назви всіх колонок і типи їхніх даних. Значення за замовчуванням (наприклад, NOT NULL) вказується за потреби в цьому користувача. Для цієї мети також може бути використано ключове слово DEFAULT. Поряд із ним вказується значення, яке може бути натуральним або дійсним числом, або NULL, але не рядковим і не NOT NULL.

Створимо таблицю «Комп’ютери» за допомогою інтерактивного SQL-редактору (рис.16.3).

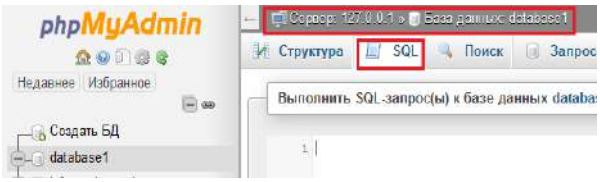


Рисунок 16.3 - Вікно SQL-редактору

У вікні редактору коду, наприклад, потрибно ввести:

```
CREATE TABLE `комп’ютери`
(`Код комп’ютера` INT NOT NULL,
`Назва` VARCHAR(20) DEFAULT 0,
`Чіпсет` VARCHAR(100) NOT NULL,
`Процесор` VARCHAR (50) NOT NULL,
`Відеоадаптер` VARCHAR (30),
`Жорсткий диск` VARCHAR (30));
```

Якщо натиснуть закладку «Структура» можна побачити результатом виконання запиту створення таблиці із наступною структурою (рис.16.4).

	#	Имя	Тип	Сравнение	Атрибуты	Null	По умолча
☐	1	**Код комп'ютера**	int(11)			Нет	*Нет*
☐	2	**Назва**	varchar(20)			Да	0
☐	3	**Чіпсет**	varchar(100)			Нет	*Нет*
☐	4	**Процесор**	varchar(50)			Нет	*Нет*
☐	5	**Відеоадаптер**	varchar(30)			Да	*NULL*
☐	6	**Жорсткий диск**	varchar(30)			Да	*NULL*

Рисунок 16.4 - Структура таблиці

Якщо потрібно створити таблицю, «знаходячись» на сервері, перший рядок запиту потрібно замінити наступним:

CREATE TABLE database1.`комп'ютери`

Для редагування визначення таблиці використовується інструкція ***ALTER TABLE.***

Наприклад, видалимо із таблиці «комп'ютери» колонку «Назва», додамо колонку «Вартість» та перейменуємо колонку «Відеоадаптер»:

```
ALTER TABLE `комп'ютери`
DROP COLUMN `Назва`,
ADD `Вартість` INT DEFAULT 0,
CHANGE `Відеоадаптер` `Відеокарта` VARCHAR(30)
```

Для перейменування таблиць використовується інструкція ***RENAME TABLE.***

Наприклад, змінимо ім'я таблиці «комп'ютери» на «pc»:

RENAME TABLE `комп'ютери` TO `pc`

Для видалення таблиць вживають інструкцію ***DROP TABLE.***

Наприклад, видалимо таблицю «pc»:

DROP TABLE `pc`

Для того, щоб зробити стовпчик ***первинним ключем,*** використовується інструкція ***ADD PRIMARY KEY.***

Наприклад, для того, щоб зробити поле «Код комп'ютера» первинним ключем потрібно виконати наступний запит:

ALTER TABLE `комп'ютери` ADD PRIMARY KEY (`Код комп'ютера`)

Запит створення первинного ключа виконається тільки у випадку, якщо в таблиці його не існує. Якщо ж він вже є, то потрібно спочатку його видалити, а потім виконувати інструкцію ADD PRIMARY KEY. За замовчуванням, якщо перший стовпчик має тип integer (тобто INT), то при створенні таблиці (якщо напряму не прописати в запиті первинний ключ) він автоматично стане первинним ключем.

Для видалення первинного ключа використовується інструкція ***DROP PRIMARY KEY:***

ALTER TABLE ` комп'ютери ` DROP PRIMARY KEY

Для того, щоб поле зробити унікальним, використовують ***інструкцію ADD UNIQUE,***

Наприклад:

ALTER TABLE `комп'ютери` ADD UNIQUE (`Відеоадаптер`)

Для того, щоб поле зробити ***індексним,*** використовується ***інструкція ADD INDEX,*** наприклад:

ALTER TABLE `комп'ютери` ADD INDEX (`Процесор`)

Встановлення зв'язків між таблицями.

Для того, щоб встановити зв'язок між таблицями, поля обох таблиць, за якими встановлюється зв'язок, повинні бути індексованими. Загальна схема запиту встановлення зв'язків між таблицями має вигляд:

```
ALTER TABLE `ім'я підрядної таблиці`
ADD FOREIGN KEY ( ` ім'я  поля підрядної таблиці, за яким встановлюється зв'язок` )
```

REFERENCES `ім’я головної таблиці ` (`ім’я  поля головної таблиці, за яким встановлюється зв’язок `)
ON DELETE RESTRICT ON UPDATE RESTRICT

16.4 Додавання, оновлення та видалення даних

Для добавляння даних у таблиці БД використовують інструкцію ***INSERT INTO***.

Для добавляння в таблицю рядка потрібно вказати назву таблиці та повний список значень для всіх полів (у тому порядку, в якому поля вказувалися при створенні таблиці), наприклад:

```
INSERT INTO `комп’ютери`
VALUES (1, 'Base NW-01', 'MSI H61M-P31/W8', 'Intel Celeron G1620', 'ASUS 210-SL-TC1GD3-L', 250)
```

Кожну інструкцію INSERT INTO слід уводити окремо.

Якщо потрібно створити декілька записів одним запитом, то в MySQL можна використати таку конструкцію:

```
INSERT INTO `комп’ютери` VALUES
(1, 'Base NW-01', 'MSI H61M-P31/W8', 'Intel Celeron G1620', 'ASUS 210-SL-TC1GD3-L', 250),
(2, 'Base MF-45', 'MSI H81M-P33', 'Intel Celeron G1820', 'Sapphire Radeon HD5450 512MB', 320),
(3, 'Universal KL-32', 'ASUS B85-PLUS', 'Intel Core i3-4330', 'ASUS GTX750-PH-1GD5', 600),
(4, 'Universal GHL-09', 'ASUS Z97-P', 'Intel Core i5-4670K', 'ASUS R7250x-2GD5', 500),
(5, 'Extreme FV-110', 'ASUS Maximus VII Ranger', 'Intel Core i5-4670K', 'ASUS R9270x-DC2T-2GD5', 1000)
```

Наведений спосіб додавання рядка в таблицю залежить від черговості слідування стовпчиків, що є не тільки незручно, але й небезпечно. Внаслідок цього, рекомендується використовувати більш громіздкий, але зручний і безпечний спосіб із вказуванням рядків, у які вставляються дані.

Наприклад,

```
INSERT INTO `комп’ютери` (`Код комп’ютера`, `Назва`, `Чіпсет`, `Процесор`, `Відеоадаптер`, `Жорсткий диск`)
VALUES (1, 'Base NW-01', 'MSI H61M-P31/W8', 'Intel Celeron G1620', 'ASUS 210-SL-TC1GD3-L', 250)
```

У цьому випадку черговість слідування значень, перерахованих в інструкції VALUES, може бути якою завгодно щодо черговості розташування стовпчиків у таблиці БД, але вона повинна відповідати черговості слідування стовпчиків у операторі INSERT INTO.

Можна додавати в таблицю ***частину рядка***, а не цілий рядок, за умови, що це дозволено у визначенні таблиці, наприклад:

```
INSERT INTO `комп’ютери` (`Назва`, `Чіпсет`)
VALUES ('Base NW-01', 'MSI H61M-P31/W8')
```

У цьому випадку значення, які відповідають невідміченим полям, заповнюються або згідно замовчувань, або приймають значення NULL, якщо останнє дозволено у визначенні таблиці (рис.16.5).

Код комп'ютера	Назва	Чіпсет	Процесор	Відеоадаптер	Жорсткий диск
1	Base NW-01	MSI H61M-P31/W8	Intel Celeron G1620	ASUS 210-SL-TC1GD3-L	250
0	Base NW-01	MSI H61M-P31/W8		*NULL*	*NULL*

Рисунок 16.5 - Результати виконання операції INSERT INTO

Для ***оновлення одного рядка*** таблиці використовується інструкція **UPDATE**.

Наприклад, змінимо розмір жорсткого диску комп'ютера із кодом 3:

```
UPDATE `комп’ютери`
SET `Жорсткий диск` = 1000
WHERE `Код комп’ютера` = 3
```

Якщо не вказати умову в операторі WHERE, то оновляться всі рядки таблиці.

Оновлення декількох рядків виконується схожим чином, наприклад, оновимо прізвище куратора із кодом 3, а також змінимо назву групи відповідної групи:

```
UPDATE `комп’ютери`
SET `Жорсткий диск` = 1000,
`Чіпсет` = ' ASUS Z97-P '
WHERE `Код комп’ютера` = 3
```

Для видалення даних із таблиці БД використовується інструкція ***DELETE FROM.***

Наприклад, видалимо із таблиці дані про куратора із кодом 3:

```
DELETE FROM `комп’ютери`
WHERE `Код комп’ютера` = 3
```

Особливістю видалення даних із таблиці є те, що видалити можна тільки весь рядок цілком, а не окремі значення. Якщо умову в інструкції WHERE пропустити, то будуть видалені всі рядки таблиці. Для ***видалення певного стовпчика*** слід використовувати інструкцію UPDATE.

16.5 Використання XAMPP для управління базою даних

XAMPP (офіційний веб-сайт **www.apachefriends.org**) — безкоштовна кросплатформова збірка веб-сервера з відкритим початковим кодом, що містить HTTP-сервер Apache, базу даних MariaDB, MySQL й інтерпретатори сценаріїв (скриптів) для мов програмування PHP та Perl, а також додаткові бібліотеки, що дозволяють запустити повноцінний веб-сервер.

(*XAMPP – це абревіатура, в якій: **X** – довільна операційна система; **A** – Apache; **M** – MySQL; **P** – PHP; **P** – Perl.)*

Для запуску XAMPP на Windows 7, потрібно знаходитись у системі під обліковим записом із повноваженнями адміністратора, або мати пароль адміністратора персонального комп’ютера. Крім цього, жодна інша із програм, наприклад, Skype не повинна займати портів 80 і 443 (якщо це сталося, то назначте цій програмі інші порти).

Якщо вказані вимоги виконано, то заходимо до меню **Пуск** і вибираємо правою кнопкою миші **Запуск від імені адміністратора** для програми **XAMPP Control Panel** (рис.16.6).

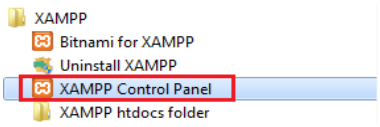


Рисунок 16.6 - Піктограма запуску XAMPP

Відкриється вікно Контрольної панелі XAMPP (рис.16.7).

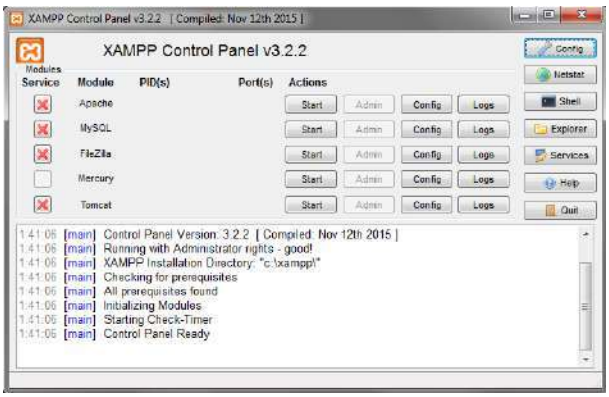


Рисунок 16.7 - Вікно Контрольної панелі XAMPP

У ньому натискаємо Start спочатку навпроти Apache, потім – навпроти MySQL (рис.16.8).

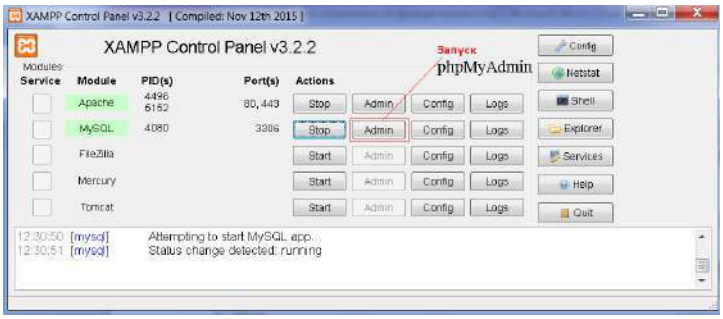


Рисунок 16.8 - Запуск модулей XAMPP

Після цього навпроти MySQL вибираємо **Admin**. У результаті цього в браузері відкриється вікно для роботи з утилітою phpMyAdmin (рис.16.9).

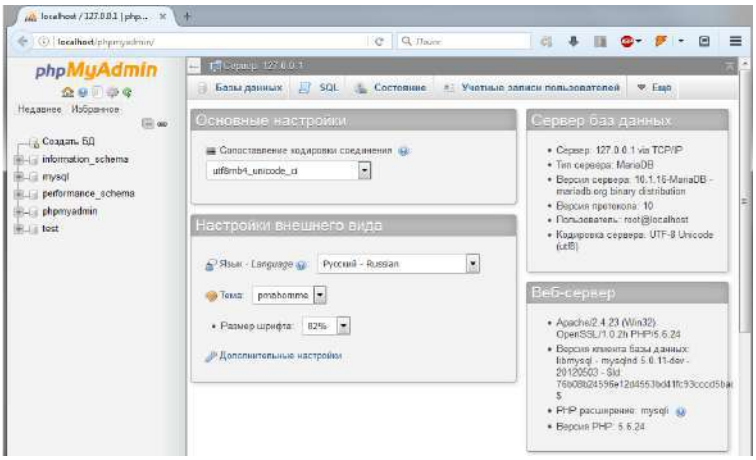


Рисунок 16.9 - Налаштування параметрів серверу

Якщо ви не хочете кожний раз при вході до Контрольної панелі запускати сервер Apache і СУБД MySQL, натисніть на Контрольній панелі **Config** і у вікні, що відкриється після цього, поставте відповідні птички навпроти **Apache** і **MySQL (рис.16.10).**

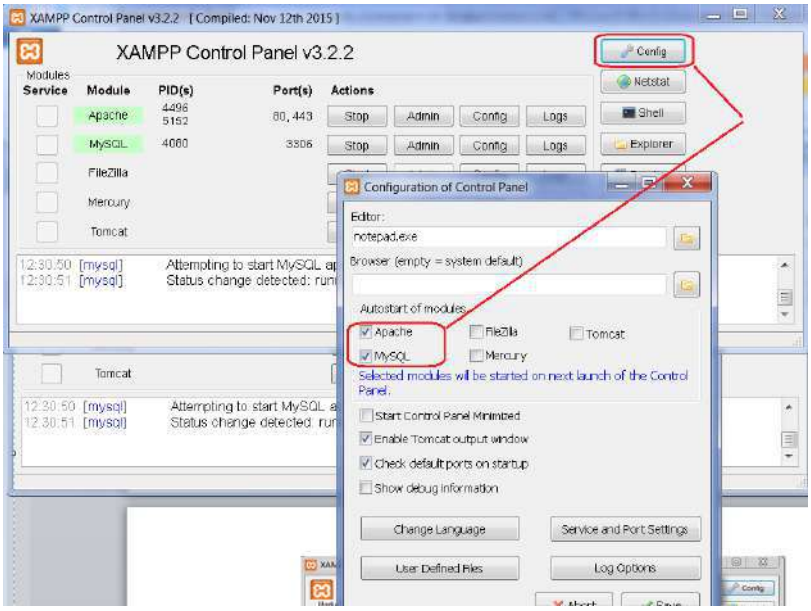


Рисунок 16.10 - Налаштування конфігурації Контрольної панелі XAMPP

Для відключення **Apache** і **MySQL** потрібно по черзі навпроти них натиснути **Stop**. Закриття Контрольної панелі не приводить до відключення зазначених сервісів.

16.6 Загальні відомості про функції MySQL

Розглянемо функції MySQL і MariaDB. Окрема увага приділена статистичним функціям. Узагалі, функції є одними з компонентів діалектів SQL, за яким останні (діалекти) найбільше відрізняються один від одного. Для зручності будемо використовувати демонстраційну таблицю. (рис.16.11).

Код комп'ютера	Назва комп'ютера	Чіпсет	Процесор	Відеоадаптер	Жорсткий диск	Вартість	Кількість	Тип
1	Base NW-01	MSI H81M-P31/W8	Intel Celeron G1620	ASUS 210-SL-TC1GD3-L	250	4000	7	Base
2	Base MF-45	MSI H81M-P33	Intel Celeron G1820	Sapphire Radeon HD5450 512MB	320	5200	4	Base
3	Universal KL-32	ASUS B85-PLUS	Intel Core i3-4330	ASUS GTX750-PH-1GD5	600	7100	5	Universal
4	Universal GHL-09	ASUS Z97-P	Intel Core i5-4670K	ASUS R7250x-2GD5	500	8100	3	Universal
5	Extreme FV-110	ASUS Maximus VII Ranger	Intel Core i5-4670K	ASUS R9270x-DC2T-2GD5	1000	9000	1	Extreme

Рисунок 16.11 - Функції MySQL і MariaDB

Функціями називають операції над даними, які використовуються дуже часто.

Суть поняття функції в MySQL є повністю аналогічною тієї, що вкладається в це поняття в мовах програмування.

Функції для роботи із текстом.

Вище була розглянута одна з таких функцій – конкатенація. Існує чимало інших функції такого типу. Наприклад, запис усіх слів рядку прописними (великими) літерами виконується за допомогою функції UPPER():

```
SELECT `Назва комп'ютера` , UPPER(`Назва комп'ютера`) AS ` Назва комп'ютера прописними літерами`
FROM `комп'ютери`
```

Результат виконання запиту на рис.16.12.

Назва комп'ютера	Назва комп'ютера прописними літерами
Base NW-01	BASE NW-01
Base MF-45	BASE MF-45
Universal KL-32	UNIVERSAL KL-32
Universal GHL-09	UNIVERSAL GHL-09
Extreme FV-110	EXTREME FV-110

Рисунок 16.12 - Функції UPPER().

Цю ж саму операцію можна провести з рядком, який не входить до таблиці бази даних, а просто записаний в інструкції SELECT:

```
SELECT UPPER ('прописні літери')
```

Результат виконання запиту:

UPPER('прописні літери')
ПРОПИСНІ ЛІТЕРИ

Функції для роботи із текстом:

LOWER() – запис усіх слів рядка малими літерами;

LENGTH() – виведення довжини рядку;

LTRIM() – видаляє пробіли в лівій частині рядку;
RTRIM() – видаляє пробіли в правій частині рядку;
SPACE(N) – виводить рядок, який складається із N пробілів;
REPEAT(рядок, N) – виводить рядок N разів.
REVERSE() – виводить рядок із оберненим порядком символів (наприклад, «мова» буде виведено як «авом»).

Функції для роботи із датами та часом:

DAYOFWEEK() – виведення дня тижня (1 – неділя, …, 7 – субота);
WEEKDAY() – виведення дня тижня (0 – понеділок, 6 – неділя);
DAYOFMONTH() – виведення дня місяця (1...31);
DAYOFYEAR() – виведення дня року (1...366);
MONTH() – виведення номера місяця;
DAYNAME() – виведення назви дня тижня (англійською);
MONTHNAME() – виведення назви місяця (англійською);
YEAR() – виведення року;
CURDATE() або CURRENT_DATE – виведення поточної дати;
CURTIME() або CURRENT_TIME – виведення поточного часу.
Крім останніх двох функцій, які записуються «як є», для всіх інших в якості аргументу треба писати дату, наприклад:
SELECT YEAR('2015-04-19')
Результат виконання запиту:

YEAR('2015-04-19')
2015

Функції для роботи із числами.

За допомогою оператора SELECT можна виконувати звичайні арифметичні операції – додавання +, множення *; віднімання -, ділення /. Порядок дій відповідає математичному пріоритету.
Наприклад,
SELECT (5+2*5)/3-1
Результат виконання запиту:

(5+2*5)/3-1
4.0000

Математичні функції одного аргументу:

– (знак мінус) – змінює знак аргументу на протилежний;
ABS() – абсолютне значення числа;
SIGN() – функція sign(x);
FLOOR(X) – найбільше ціле число, яке не перевищує X;
CEILING(X) – найменше ціле число, не менше, ніж X;
ROUND() – округлення;
ROUND(X,D) – аргумент X, округлений до числа із D десятковими знаками;
EXP() – експонента;
LN() – натуральний логарифм;
LOG(B, X) – логарифм числа X за основою B;
LOG2(X) – логарифм числа X за основою 2 (використовується для з’ясування того, скільки бітів потребує число для його зберігання);
LOG10(X) – десятковий логарифм;

POW(X,Y), POWER(X,Y) – число X, піднесене у степінь Y;
SQRT(X) – квадратний корінь числа X;
PI() – число π;
COS(X), SIN(X), TAN(X), COT(X) – відповідно косинус, синус, тангенс числа та котангенс числа X, яке задається в радіанах;
ACOS(X), ASIN(X), ATAN(X) – відповідно арккосинус, арксинус та арктангенс числа;
RADIANS(X) – перетворення з радіанів у градуси;
Приклади із результатами їх виконання:
SELECT (5+2*5)/3-1

(5+2*5)/3-1
4.0000

SELECT ROUND((5+2*5)/3-1,2)

ROUND((5+2*5)/3-1,2)
4.00

SELECT ROUND((5+2*5)/3-1)

ROUND((5+2*5)/3-1)
4

SELECT PI()

PI()
3.141593

SELECT ROUND(PI(),10)

ROUND(PI(),10)
3.1415926536

SELECT COS(PI())

COS(PI())
-1

16.7 Статистичні обчислення та групування даних

Статистичні функції:
AVG() – середнє значення стовпчика;
COUNT() – кількість рядків у стовпчику;
MAX() – найбільше значення у стовпчику;
MIN() – найменше значення у стовпчику;
SUM() – сума значень стовпчика.
Приклад:
SELECT COUNT(`Кількість`) AS 'Кільк. комп.', SUM(`Вартість`) AS 'Загальна вартість комп.', MAX(`Вартість`) AS 'Вартість найдорож. комп.', MIN(`Вартість`) AS 'Вартість найдеш. комп.', AVG(`Вартість`) AS 'Середня вартість комп.'
FROM `комп'ютери`
Результат виконання запиту:

Кільк. комп.	Загальна вартість комп.	Вартість найдорож. комп.	Вартість найдеш. комп.	Середня вартість комп.
5	33400	9000	4000	6680.0000

Зауваження щодо використання статистичних функцій:

- Стовпчики зі значенням NULL ігноруються функціями MAX, MIN та SUM.

- Функція COUNT() у вигляді COUNT(*) підраховує кількість рядків у таблиці, незалежно від того, містить ця таблиця стовпчики із значенням NULL чи ні.
- Функцію AVG() можна використовувати тільки для обчислення середнього значення певного числового стовпчика. Для обчислення середнього значення декількох стовпчиків треба використовувати декілька функцій AVG().

Групування і фільтрація даних у статистичних обчисленнях.

Для групування даних у статистичних обчисленнях використовується речення GROUP BY. Наприклад, знайдемо середню вартість комп'ютерів для кожного типу:

```
SELECT `Тип`, AVG(`Вартість`) AS 'Середня вартість'
FROM `комп’ютери`
GROUP BY `Тип`
```

Результат виконання запиту:

Тип	Середня вартість
Base	4600.0000
Extreme	9000.0000
Universal	7600.0000

Групувати результати можна і за декількома стовпчиками.

Речення GROUP BY повинен іти після WHERE та перед ORDER BY.

Для фільтрації даних, згрупованих за допомогою GROUP BY, використовується оператор HAVING. Наприклад, середню вартість комп'ютерів для кожного типу, виключаючи ті типи, для яких вона більша за 8000:

```
SELECT `Тип`, AVG(`Вартість`) AS 'Середня вартість'
FROM `комп’ютери`
GROUP BY `Тип`
HAVING AVG(`Вартість`)<8000
```

Результат виконання запиту:

Тип	Середня вартість
Base	4600.0000
Universal	7600.0000

Зауваження щодо групування і фільтрації даних у статистичних обчисленнях:

- Для гарантовано правильного сортування даних оператором GROUP BY поряд із ним слід писати речення ORDER BY.
- Речення WHERE призначене для фільтрації незгрупованих даних, HAVING – для згрупованих.
- Стовпчики, до яких застосовується речення HAVING, повинні бути обов’язково записані в інструкції SELECT, тобто бути вибраними. Для використання речення HAVING груп повинно бути не менше двох.
- У реченні HAVING допускається використання метасимволів і логічних операторів так само, як у WHERE.

Контрольні питання.

1. Для чого призначені статистичні функції?
2. Які використовуються математичні функції одного аргументу?
3. Як виконується функції для роботи із текстом ?
4. Як виконується фільтрація записів ?
5. Як виконується пошук записів при зовнішньому з’єднанні відношень ?

17. СУБД IBM DB2

17.1 Склад серверу DB2. Структура бази даних

Сервер даних IBM DB2 Express-C 9 являє собою повно-функціональну систему керування базами даних, що підтримує роботу з даними в форматі XML. Інформація, що зберігається, може бути відображена у клієнтських додатках що створені в Java, .NET, PHP і Ruby on Rails. Включає в себе можливості самоврядування і автоматичної оптимізації продуктивності. Підтримує кластеризації з високим ступенем доступності та реплікацію даних. Простий в установці, впровадженні і розгортанні.

СУБД DB2 включає різні об'єкти бази даних і файли конфігурації. На рис.17.1 представлений огляд різних команд і інструментів для роботи з DB2, а в правій частині виділено середовище DB2. У лівій частині малюнка показані різні команди DB2, а також оператори SQL, SQL/XML і XQuery, використовувані для взаємодії з сервером даних DB2. У центральній частині малюнка відображені імена різних інструментів, використовуваних для взаємодії з сервером даних DB2.

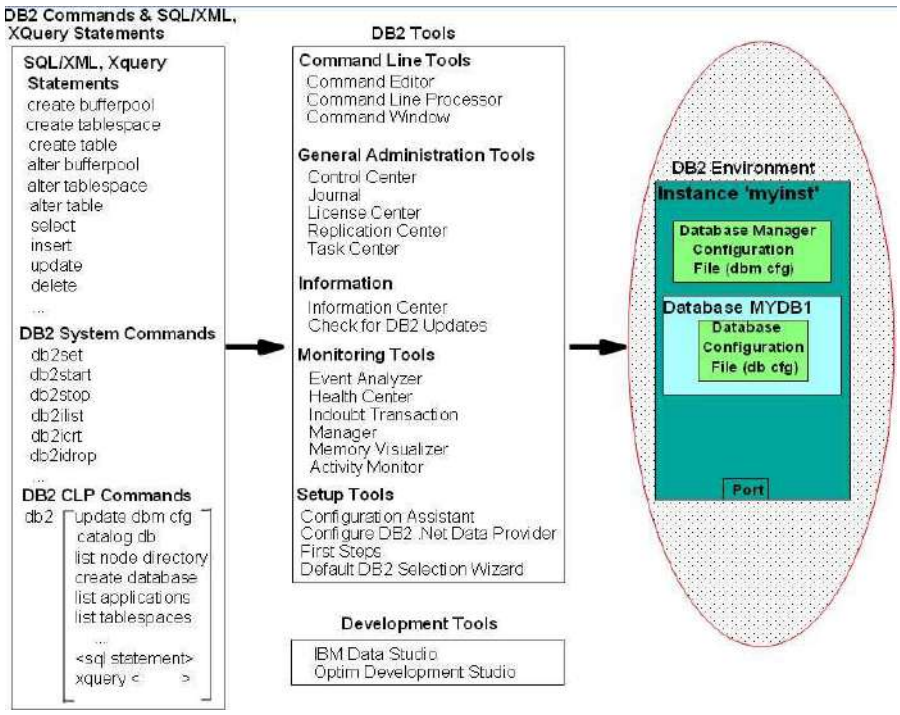


Рисунок 17.1 - Загальний огляд DB2

В процесі установки в Windows за умовчанням створюється екземпляр з ім'ям DB2. На рис.17.2 йому відповідає зелений блок ліворуч. **Екземпляр** - це незалежне середовище, в якому можуть працювати додатки і створюватися бази даних. На сервері даних можна створити декілька екземплярів і використати їх в різних цілях. Приміром, один екземпляр можна використати для зберігання виробничих баз даних, інший - баз даних середовища тестування, ще один може використовуватися як середовище розробки. Кожен екземпляр є незалежним, т. е. виконання операцій в одному з них не впливає на інші.

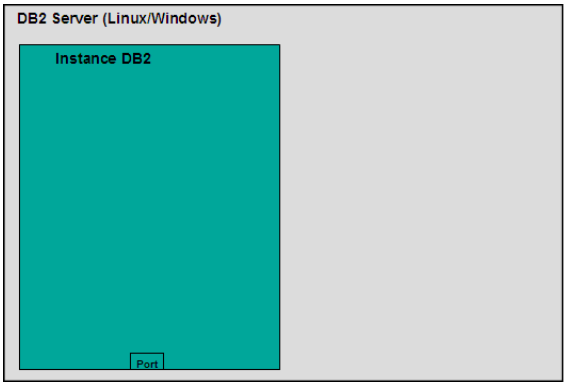


Рисунок 17.2 - Екземпляр DB2, створений за умовчанням

Для створення нового екземпляра DB2 використовують команду **db2icrt <ім'я екземпляра>**. Ім'я повинно складатися не більш восьми символів. Наприклад, для створення екземпляра **myinst** використовується команда **db2icrt myinst.** На рис. 17.3 новий екземпляр з ім'ям **myinst** показаний як окремий зелений блок справа.

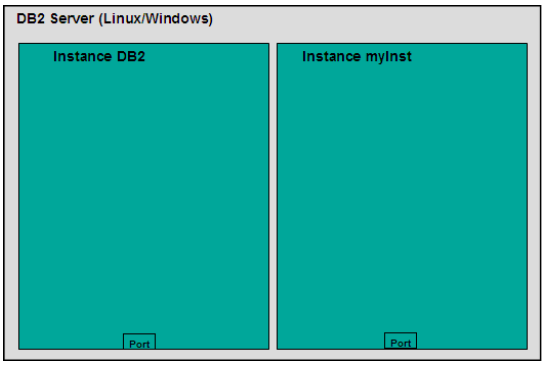


Рисунок 17.3 - Сервер DB2 з двома екземплярами

Кожен екземпляр має унікальний номер порту. Це допомагає розрізняти екземпляри, якщо необхідно підключитися до екземпляра бази даних з видаленого клієнта по TCP/IP. Якщо використовується командне вікно DB2, можна будь-який з екземплярів DB2 зробити активним, виконавши наступну команду операційної системи Windows: **set db2instance=myinst , з** відсутністю пропусків до і після знаку рівності (=). Якщо в даному прикладі створити базу даних в командному вікні, вона буде створена в екземплярі **myinst.** Щоб вивести список усіх екземплярів системи, виконаєте команду: **db2ilist.** У таблиці 17.1 наведені деякі корисні команди рівня екземпляру.

Таблиця 17.1 Корисні команди DB2 на рівні екземпляру

Команда	Опис
db2start	Запустити поточний екземпляр
db2stop	Зупинити поточний екземпляр
db2icrt	Створити новий екземпляр
db2idrop	Видалити екземпляр
db2ilist	Показати список примірників в системі
db2 get instance	Показати поточний активний екземпляр

Перерахованих вище команд можна також виконати через Центр управління. Наприклад, якщо в Центрі управління розкрити папку Примірники і клацнути правою кнопкою миші по необхідному примірнику, можна вибрати пункт Запуск, що відповідає виконанню команди db2start в командному вікні DB2, або Зупинити, що відповідає команді **db2stop** (рис. 17.4).

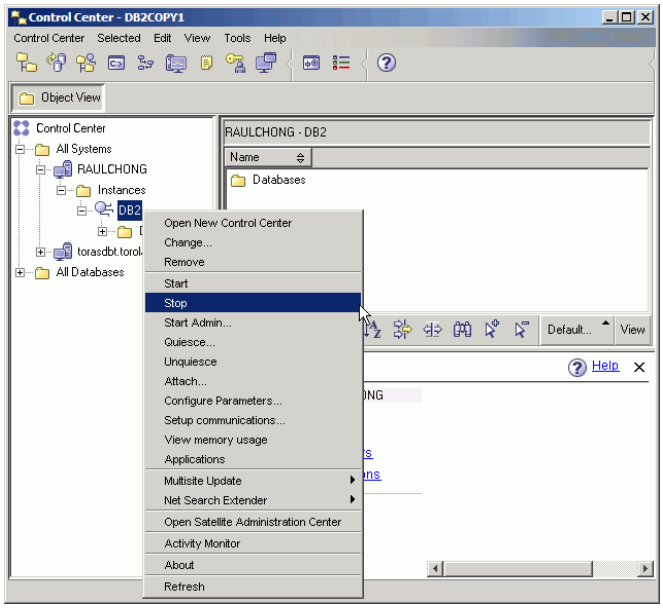


Рисунок 17.4 - Команди екземпляра в Центре управління

Щоб створити базу даних в активному екземплярі, в командному вікні DB2 виконайте команду: **db2 create database mydb1**. Щоб перелічити всі створені бази даних, виконайте команду: **db2 list db directory.**

В рамках одного примірника можна створити багато баз даних. **База даних** - це сікупність таких об'єктів, як таблиці, уявлення, індекси тощо. Бази даних є незалежними одиницями і, відповідно, не використовують об'єкти спільно з іншими базами даних (рис. 17.5).

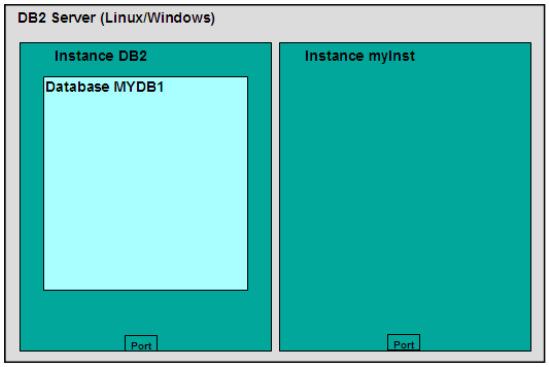


Рисунок 17.5 - База даних MYDB1, яка створена в екземплярі DB2

У таблиці 17.2 описані деякі команди, які можна використовувати на рівні баз даних.

Таблиця 17.2 Команди SQL-оператори на рівні баз даних

Команда/SQL-оператор	Опис
db2 create database	Створити нову базу даних
db2 drop database	Видалити базу даних
db2 connect to <имя_базы_данных>	Підключитися до бази даних
db2 create table/create view/create index	Оператори SQL для створення таблиць, уявлень і індексів, відповідно

Якщо потрібно створити ще одну базу даних з таким же ім'ям (MYDB1), але в екземплярі myinst, необхідно виконати наступні команди в командному вікні DB2:

db2 list db directory set db2instance = myinst

db2 create database mydb1 set db2instance = db2

На рис. 17.6 показана нова база даних MYDB1, створена в екземплярі myinst.

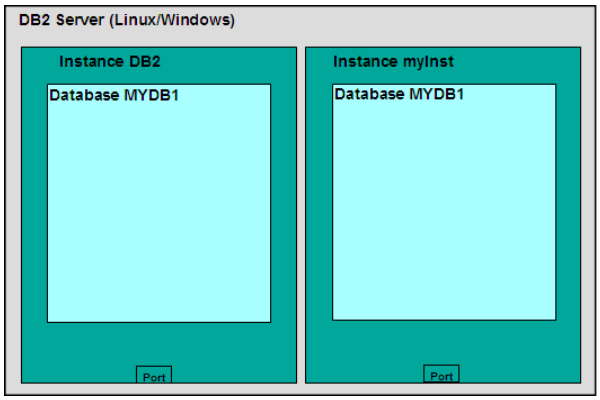


Рисунок 17.6 - База даних MYDB1, створена в екземплярі myInst

При створенні бази даних кілька об'єктів створюються за замовчуванням:

- табличні простору,
- таблиці,
- буферний пул,
- файли журналу.

Створення цих об'єктів займає деякий час, тому команда create database виконується кілька хвилин. У лівій частині рис. 17.7 показані три табличних простору, створені за замовчуванням. На даному етапі будемо розглядати табличні простору як логічний шар між логічними таблицями і фізичними ресурсами, такими як диски і пам'ять.

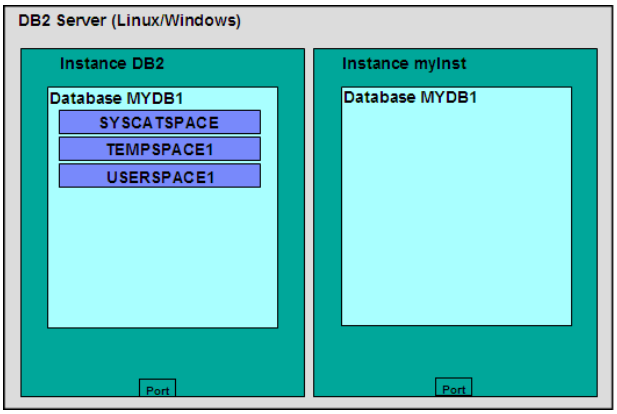


Рисунок 17.7 - Табличні просторі, створені за замовчуванням

Табличний простір SYSCATSPACE містить таблиці системного каталогу. В інших реляційних СУБД системний каталог називається словником даних. По суті, він містить системну інформацію, яку не можна змінювати або видаляти, інакше база даних не зможе коректно працювати.

Табличний простір TEMPSPACE1 використовується в DB2, коли потрібно додатковий простір для виконання деяких операцій, таких як сортування.

Табличний простір USERSPACE1 зазвичай використовується для зберігання призначених для користувача таблиць бази даних, якщо при створенні таблиці не вказано інше табличний простір.

Також можна створити власні табличні простору, скориставшись оператором CREATE TABLESPACE. На рис. 17.8 показано табличний простір MYTBLS1, якій створене в базі даних MYDB1 в екземплярі DB2. При створенні табличного простору вказують, які диски і пам'ять (буферний пул) будуть використовуватися. На рис.17.9 показані ще два об'єкти, що створюються за замовчуванням: буферний пул IBMDEFAULTBP і

файли журналу.

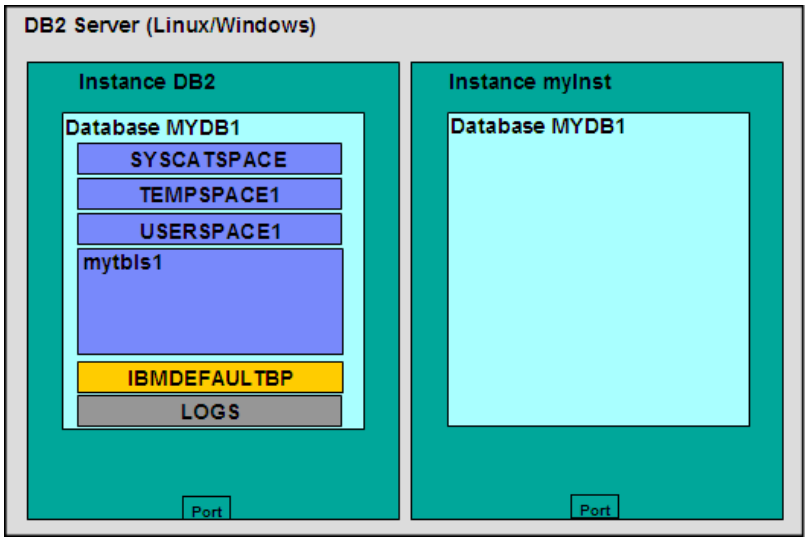


Рисунок 17. 8 - Буферний пул і журнали, створені за замовчуванням

Буферний пул - це, по суті, кеш-пам'ять, використовувана базою даних. Можна створити один або кілька буферних пулів, але обов'язково повинен існувати хоча б один буферний пул, розмір сторінки якого відповідає розміру сторінки існуючих табличних просторів.

Файли журналу використаються для відновлення. Під час роботи з базою даних на відповідних дисках зберігається інформація, але крім цього всі виконувані з даними операції зберігаються у файлах журналу. Журнали можна розглядати як тимчасові файли, у яких виконується «автоматичне збереження».

Екземпляри є незалежними середовищами, і тому в декількох екземплярах можна створити бази даних з однаковим ім'ям. Як і екземпляри, бази даних також є незалежними одиницями; відповідно, об'єкти однієї бази даних не мають ніякого відношення до об'єктів іншої. На рис. 17.9 показане табличний простір ***mytbls1*** у базах даних ***MYDB1*** і **SAMPLE** в екземплярі **DB2**. Таке можливо, оскільки бази даних є незалежними одиницями.

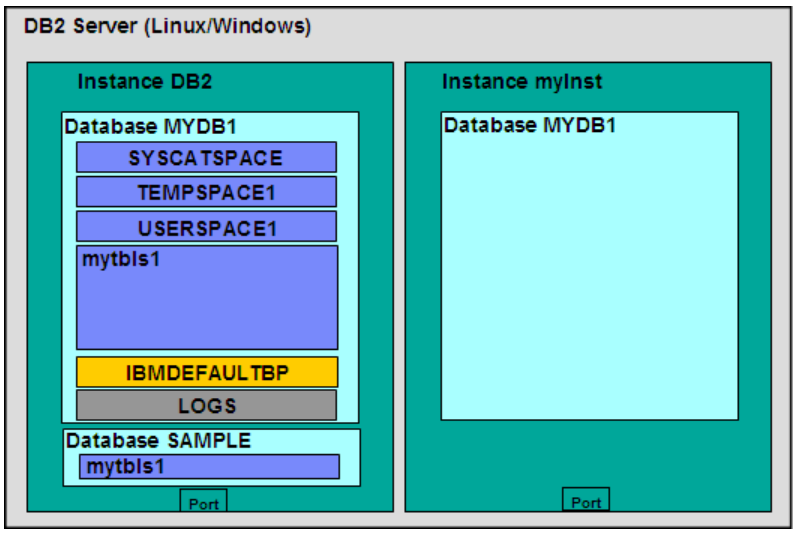


Рисунок 17.9 - Табличні простори з однаковим ім'ям у різних базах даних.

Створивши табличний простір, можна створити в ньому такі об'єкти, як таблиці, подання й індекси (рис. 17.10).

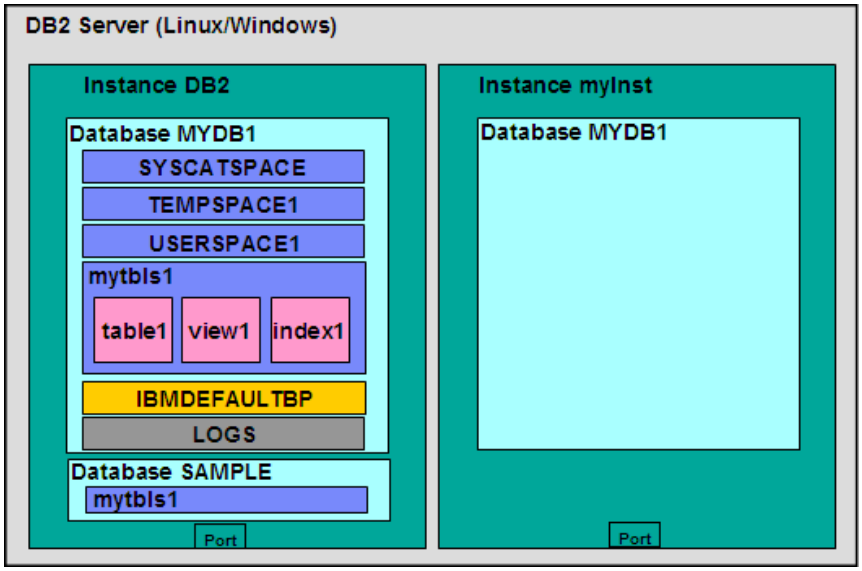


Рисунок 17.10 - Таблиці, подання й індекси, створені в табличному просторі

17.2 Налаштування конфігурації серверу DB2

Параметри DB2 можна настроїти за допомогою інструмента «Порадник по конфігуруванню». Щоб відкрити порадник по конфігуруванню із Центра керування, клацніть правою кнопкою миші базу даних і виберіть пункт *Порадник по конфігуруванню*. Залежно від відповідей на деякі питання про системні ресурси й робоче навантаження порадник по конфігуруванню надасть список параметрів DB2, які варто було б змінити, а також укаже кращі значення для кожного з параметрів.

Конфігурація сервера DB2 може бути задана на чотирьох різних рівнях (рис.17.11):

- змінні середовища;
- файл конфігурації менеджера бази даних (dbm cfg);
- файл конфігурації бази даних (db cfg);
- реєстр профілю DB2.

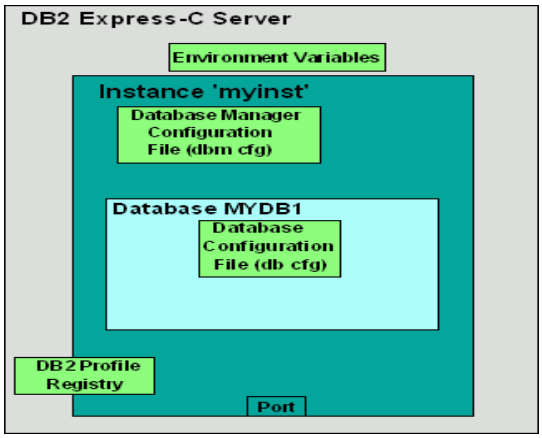


Рисунок 17.11- Конфігурування DB2

Змінні середовища задаються на рівні операційної системи сервера, а параметри файлу конфігурації менеджера бази даних - на рівні екземпляра. Параметри конфігурації бази даних задаються на рівні бази даних, а реєстр профілю DB2 - на рівні операційної системи або екземпляра.

Змінні середовища — це змінні, установлені на рівні операційної системи. Одна із ключових змінні середовища — **DB2INSTANCE**. Ця змінна позначає активний екземпляр, у якому виконується робота й до якого будуть застосовуватися команди DB2. Приміром, щоб установити ***myinst*** як активний екземпляр, в Windows можна виконати наступну команду операційної системи: **set db2instance=myinst**

Файл конфігурації менеджера бази даних (dbm cfg) включає параметри, що впливають на екземпляр і всі бази даних, що втримуються в ньому. Файл конфігурації менеджера бази даних можна переглянути або змінити за допомогою командного рядка або через Центр керування DB2.

Для роботи з dbm cfg через Центр керування виберіть об'єкт екземпляра в папці екземплярів центра керування, клацніть правою кнопкою миші й у спливаючому меню виберіть пункт *Конфігурувати параметри (*рис.17. 12*)*.

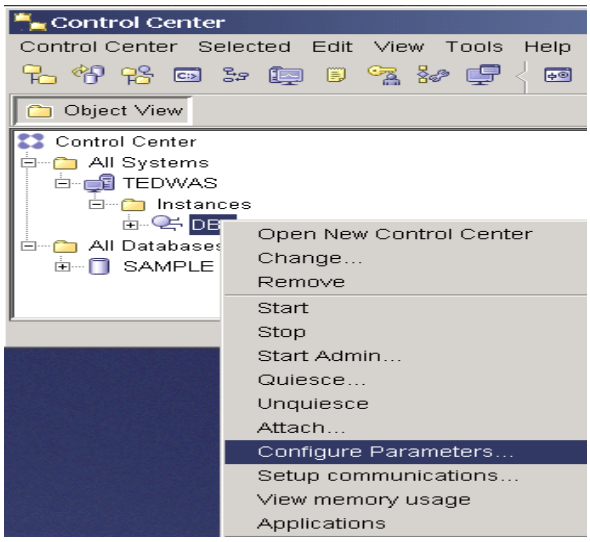


Рисунок 17.12 - Конфігурування dbm cfg із Центра керування

Після вибору пункту *Конфігурувати параметри* відкриється зображене на рис. 17.13 вікно зі списком параметрів dbm cfg.

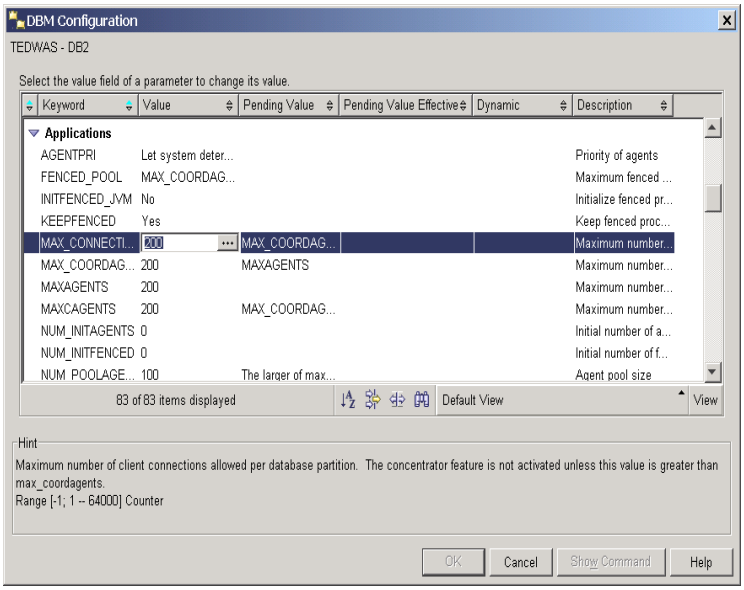


Рисунок 17.13 - Діалогове вікно dbm cfg

Багато параметрів є динамічними, тобто внесені зміни відразу ж набувають чинності; однак є параметри, для зміни яких необхідно зупинити й знову запустити екземпляр. Це можна зробити в командному рядку за допомогою команд **db2stop** і **db2start**.

Перед зупинкою екземпляра всі додатки повинні відключитися. Для примусової зу-

пинки екземпляра можна скористатися командою **db2stop force**.

Екземпляр також можна зупинити через Центр керування, клацнувши на об'єкт екземпляра й вибравши пункт *Зупинити* або *Запуск*. У таблиці 17.3 представлені деякі корисні команди для керування dbm cfg з командного рядка.

Таблиця 17.3. Команди для роботи з dbm cfg

Команда	Опис
db2 get dbm cfg	Витяг інформації про dbm cfg
db2 update dbm cfg using <ім'я_параметра> <значення>	Відновлення значення параметра dbm cfg

Файл конфігурації бази даних (db cfg) включає параметри, що впливають на певну базу даних. Файл конфігурації бази даних можна переглянути або змінити за допомогою командного рядка або через Центр керування DB2.

Для роботи з db cfg через Центр керування виберіть об'єкт бази даних у папці баз даних центра керування, клацніть правою кнопкою миші й у спливаючому меню виберіть пункт *Конфігурувати параметри*. Це показано на рис. 17.14.

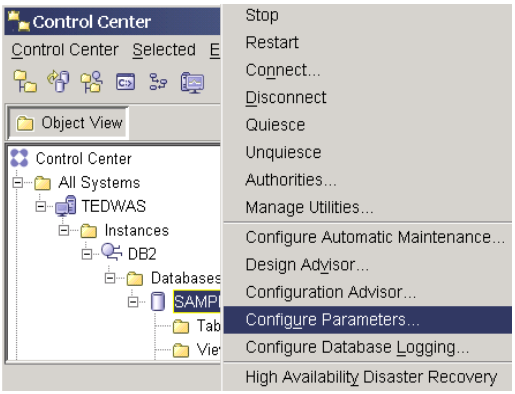


Рисунок 17.14 - Конфігурування db cfg через Центр керування

Після вибору пункту *Конфігурувати параметри* відкриється зображене на рис.17. 15 вікно зі списком параметрів db cfg.

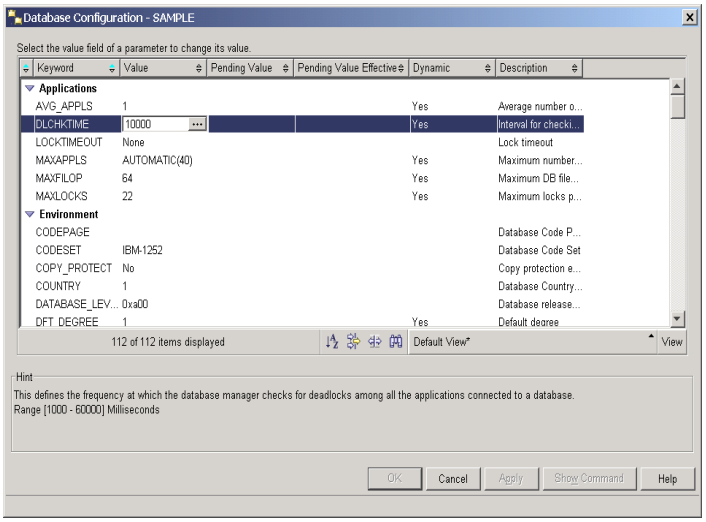


Рисунок 17.15 - Список db cfg

У таблиці 4 представлені деякі корисні команди для керування db cfg з командного рядка.

Таблиця 17.4. Команди для роботи з db cfg.

Команда	Опис
get db cfg for <ім'я_бази_даних>	Витяг інформації про db cfg
update db cfg for <ім'я_бази_даних> using <ім'я_параметра> <значення>	Відновлення значення параметра db cfg

Реєстр профілю DB2. Змінні реєстру профілю DB2 включають параметри, які можуть залежати від платформи й установлюватися глобально (для всіх екземплярів) або на рівні екземпляра (для одного певного екземпляра). У таблиці 17.5 представлені деякі корисні команди для керування реєстром профілю DB2.

Таблиця 17. 5 Команди для керування реєстром профілю DB2.

Команда	Опис
db2set -all	Висновок списку всіх заданих змінних реєстру профілю DB2
db2set -lr	Висновок списку всіх змінних реєстру профілю DB2
db2set <параметр>=<значення>	Присвоювання параметру заданого значення

Таблиця 17.6 Змінні реєстру DB2

Змінна реєстру	Опис
DB2COMM	Визначає, які менеджери передачі даних запускаються при запуску менеджера бази даних
DB2_EXTSECURITY	Для Windows: запобігає несанкціонованому доступу до DB2, блокуючи системні файли DB2
DB2_COPY_NAME	Зберігає ім'я використовуваної тепер копії DB2. Щоб перемкнутися на іншу встановлену копію DB2, виконаєте команду **installpath\bin\db2envar.bat**

Щоб дозволити з'єднання через протокол TCP/IP, установите для змінної реєстру DB2COMM значення TCPIP, як показано нижче: **db2set db2comm=tcpip**

17.3 Створення таблиць баз даних у Центрі управління

З вікна Центра керування DB2 створімо таблицю Artists, з наступними характеристиками (таблиця 17.7):

Таблиця 17.7 Характеристики нової таблиці

Ім'я стовпця	Тип даних	Довжина	Значення NULL	LOB OPTION
artno	smallint		No	
name	varchar	50	Yes	
classification	character	1	No	
bio	clob	100 KBytes	Yes	Logged, Compact
picture	blob	500 KBytes	Yes	Compact

Відкрийте Центр керування DB2, використовуючи Розширений (Advanced) вид (рис.17.16).

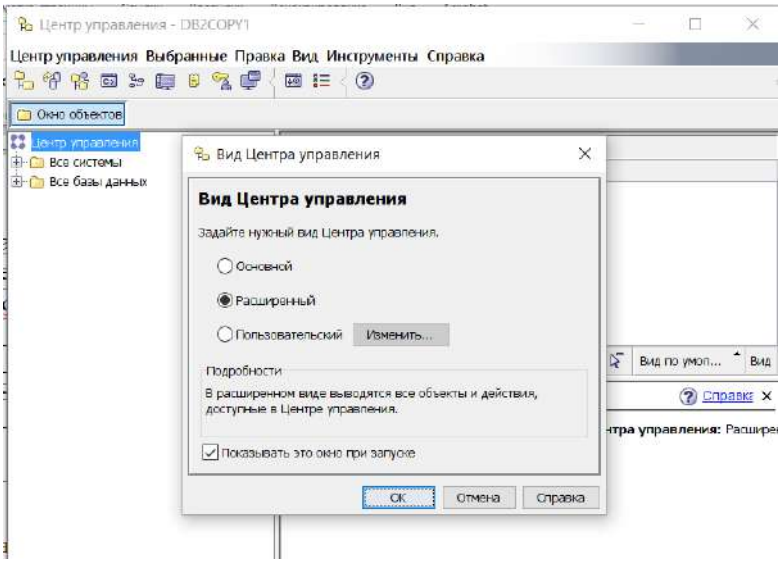


Рисунок 17.16 - Центр керування DB2

Виконаєте команду **db2start,** для чого викличте правою кнопкою контексное меню й виберіть рядок ***Запуск*** (рис.17.17).

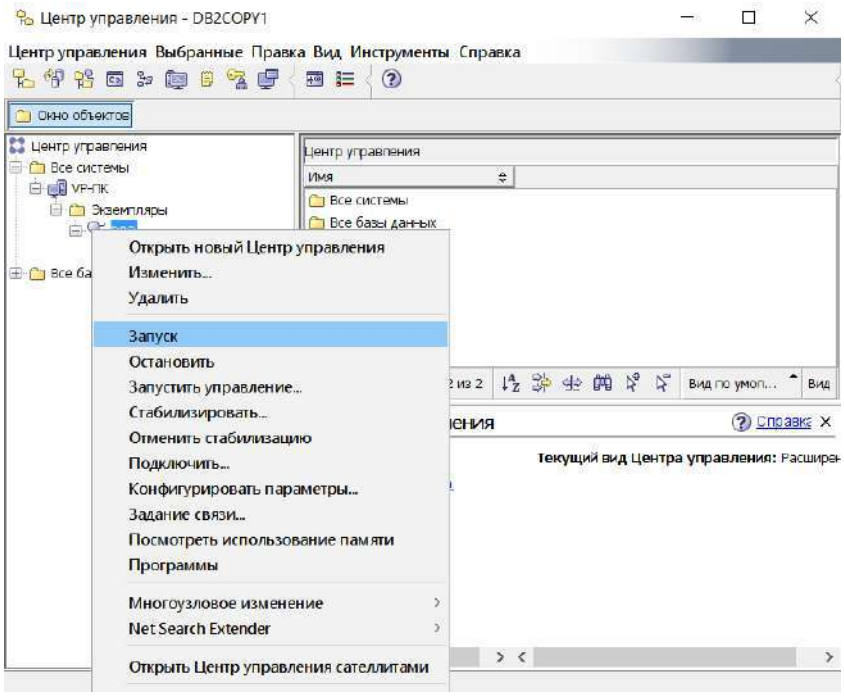


Рисунок 17.17 - Запуск серверу

Створіть нову схему БД можна одним із двох способів.

Перший спосіб: вибір команди ***Створити базу даних*** з контекстного меню (рис.17.18).

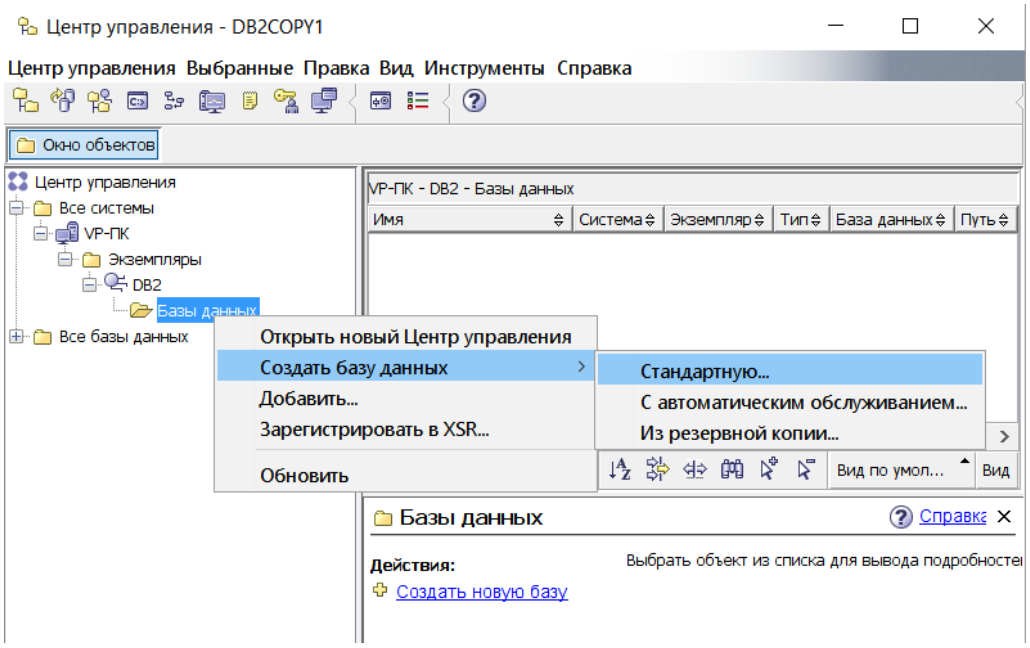


Рисунок 17.18 - Створення БД

Другий спосіб створення БД: натисніть посилання в розділі Бази даних ***Створити нову базу*** (рис.17.19).

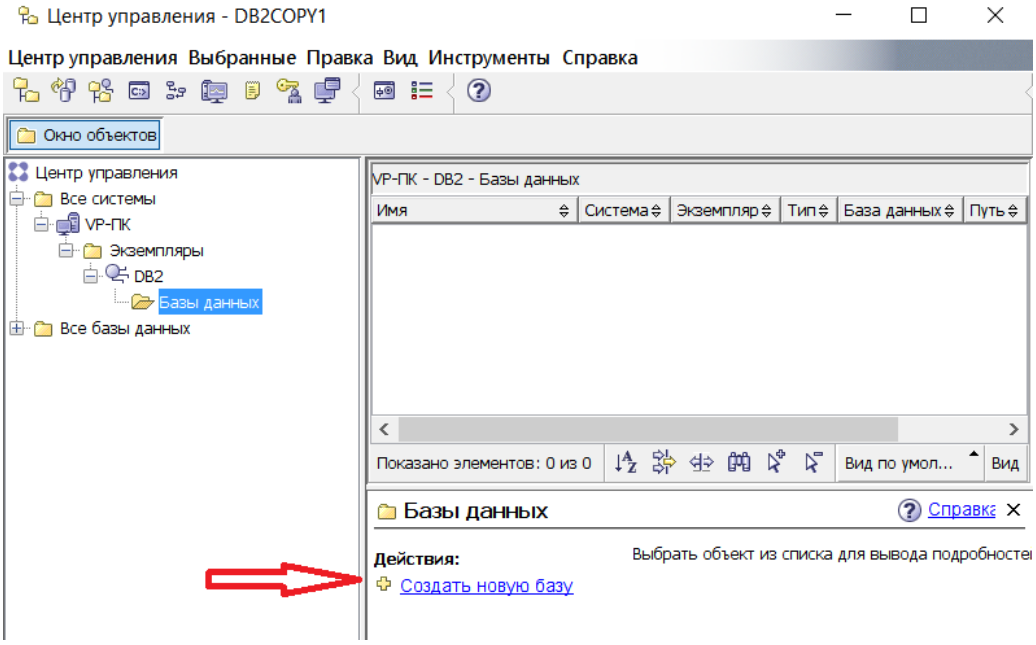


Рисунок 17.19 - Спосіб створення БД з вікна каталогу

Задайте ім'я БД - MUSICDB. Укажіть шлях за замовчуванням , наприклад - C:\.
Задайте Алиас БД - MUSICDB. Натисніть кнопку Далі (рис.17.20).

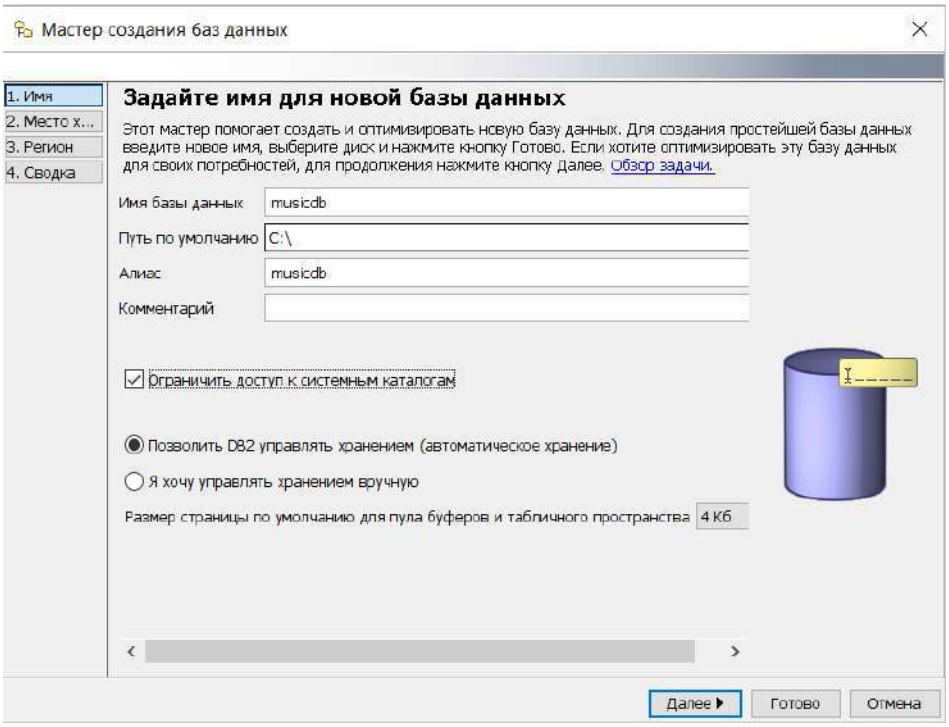


Рисунок 17.20 - Налаштування параметрів БД

У вікні «Задайте місце зберігання ваших даних», можете вказати шлях зберігання, Вільне місце БД і Ємність файлів БД. Натисніть кнопку «Далі» (рис.17.21).

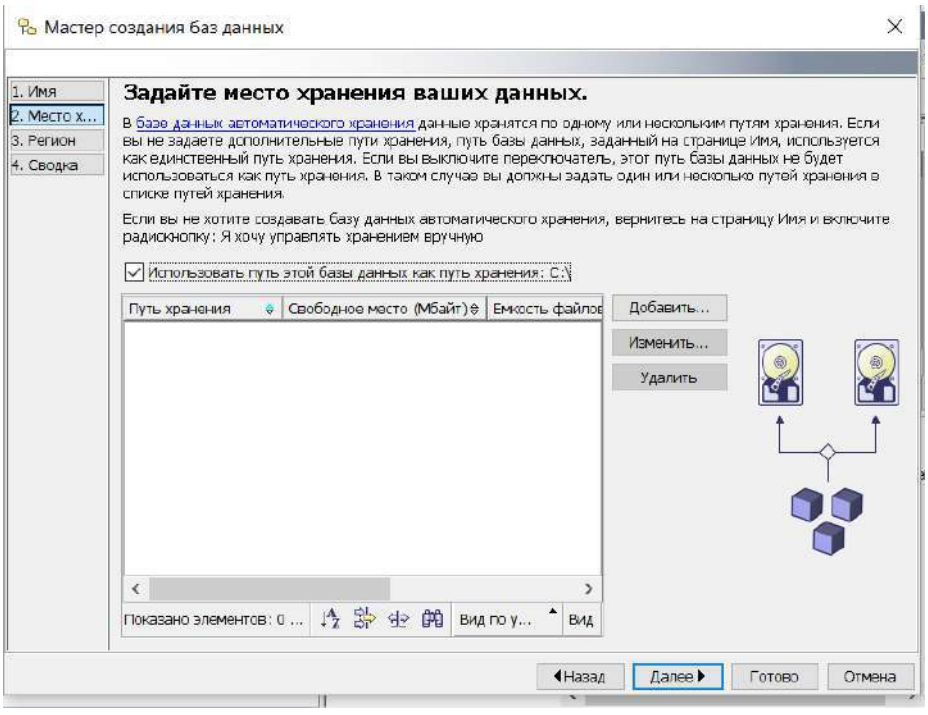


Рисунок 17.21 - Налаштування місця збереження файлів БД

У новому вікні задайте національну версію для цієї бази даних як показано на малюнку нижче. Натисніть кнопку «Далі» (рис.17.22).

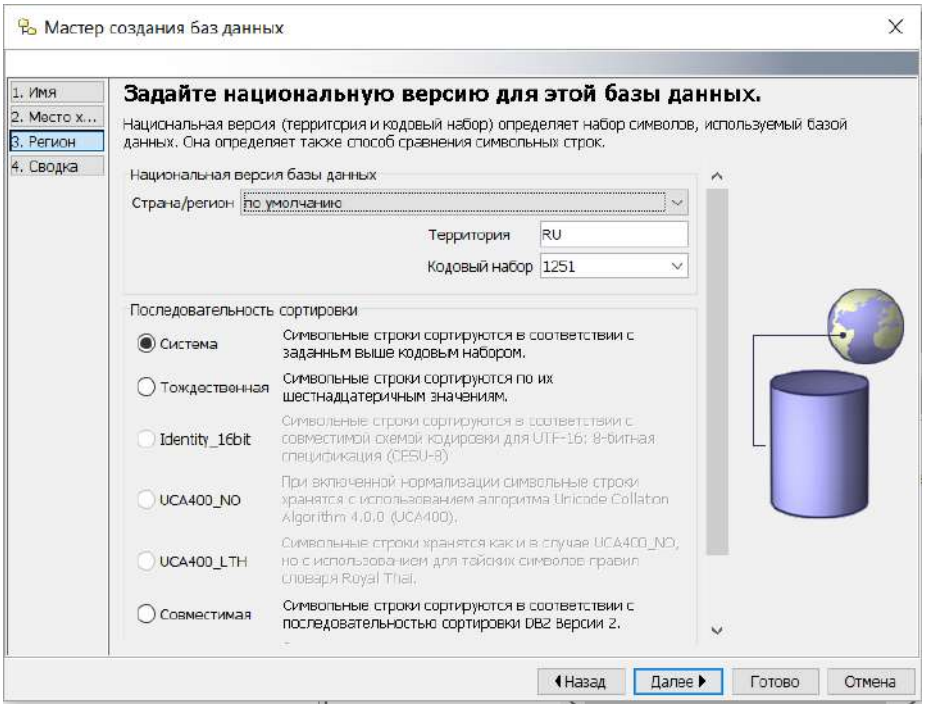


Рисунок 17.22 - Налаштування коду кодування

Перевірте протокол і натисніть кнопку Готово (рис.17.23).

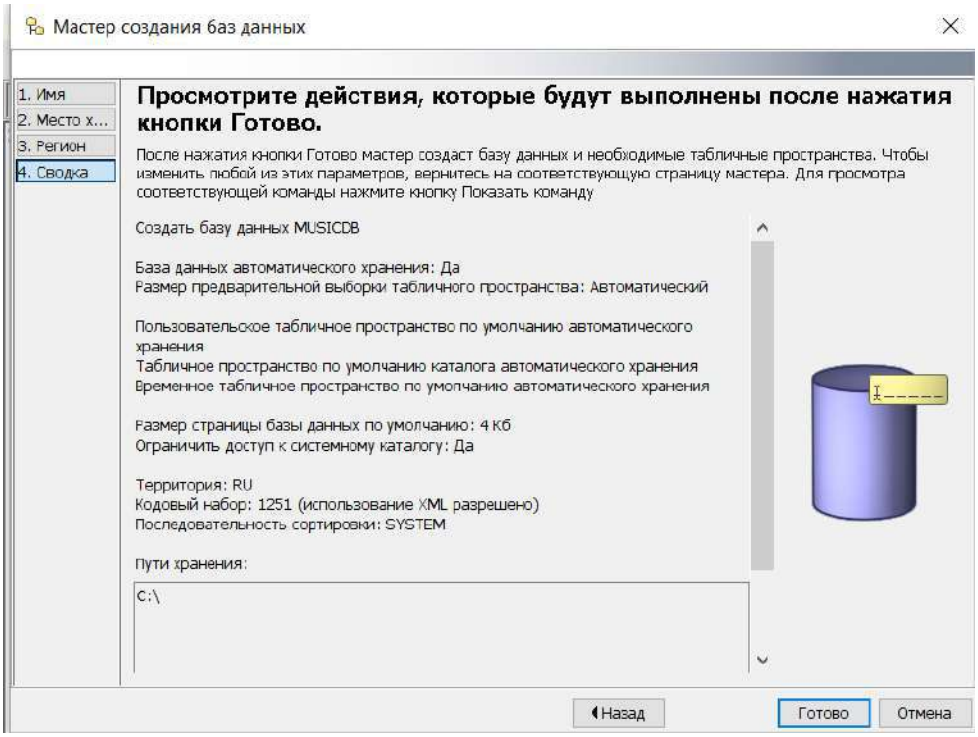


Рисунок 17.23 - Вікно перевірки параметрів

На протязі 2-3 хвилин СУБД створить всі необхідні об'єкти та видасть повідомлення про помилки або успішне виконання (рис.17.24).

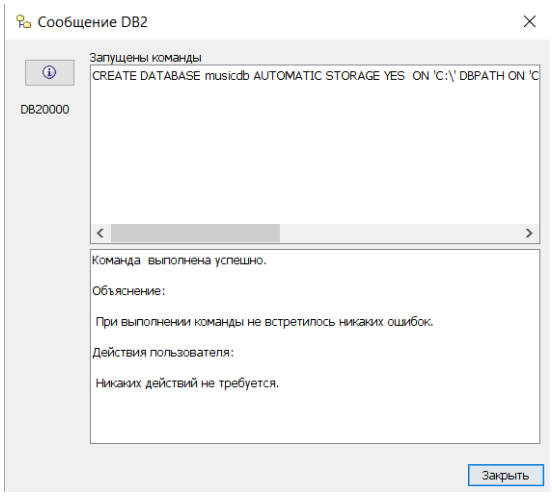


Рисунок 17.24 - Вікно повідомлень о результаті налаштування БД

Розкрийте вміст папки бази даних MUSICDB (рис.17.25).

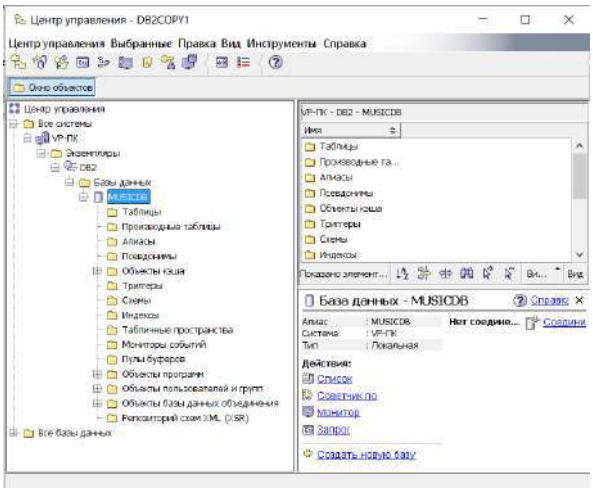


Рисунок 17.25 - Склад нової БД

У списку об'єктів бази даних MUSICDB клацніть правою кнопкою миші на Таблиці (Tables) і виберіть Створити (Create) зі спливаючі меню (рис.17.26).

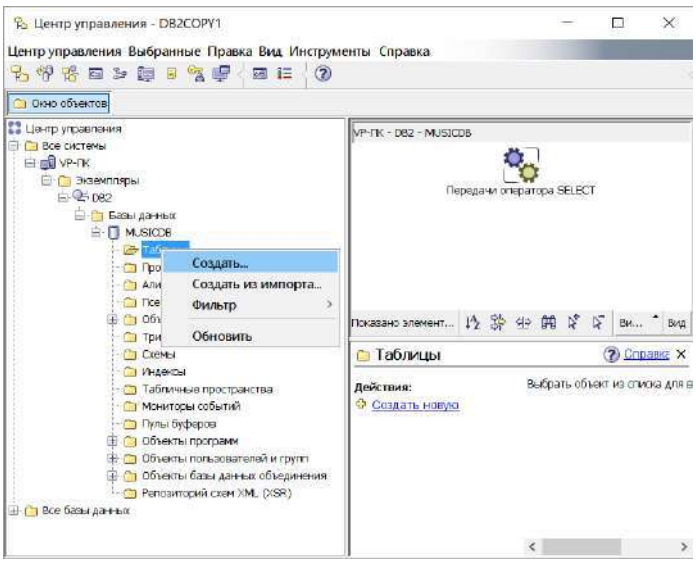


Рисунок 17.26 - Вікно для створення таблиці БД

Виберіть ім’я схеми таблиці БД зі списку, що розкривається, поля Схема таблиці (Table schema).

Уведіть значення Artists у поле Ім'я таблиці (Table name). Натисніть Далі (Next) для того щоб визначити стовпці таблиці (рис.17.27).

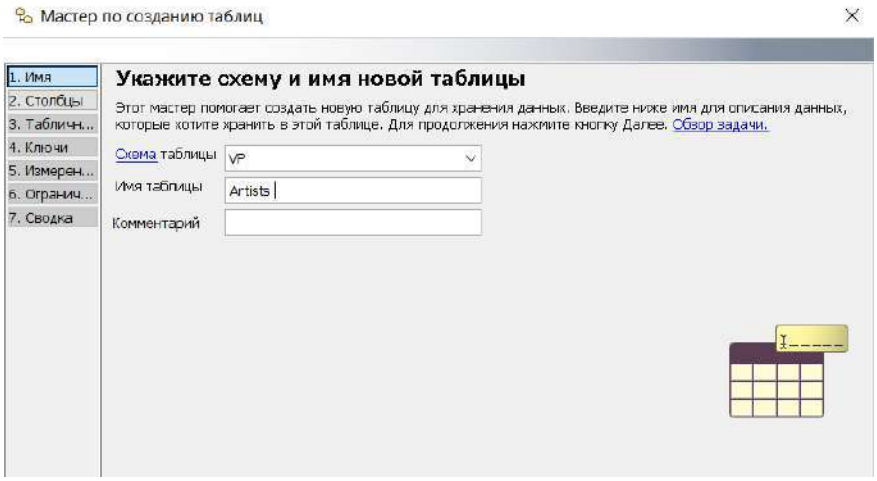


Рисунок 17.27 - Реєстрація параметрів аблиці

У вікні Стовпці (Columns) визначите стовпець artnom (рис.17.28):

- Натисніть кнопку Додати (Add).
- Уведіть artno у поле Ім'я стовпця (Column name).
- Виберіть SMALLINT зі списку, що розкривається, поля Тип даних (Data type).
- Натисніть кнопку Застосувати (Apply), потім ОК.

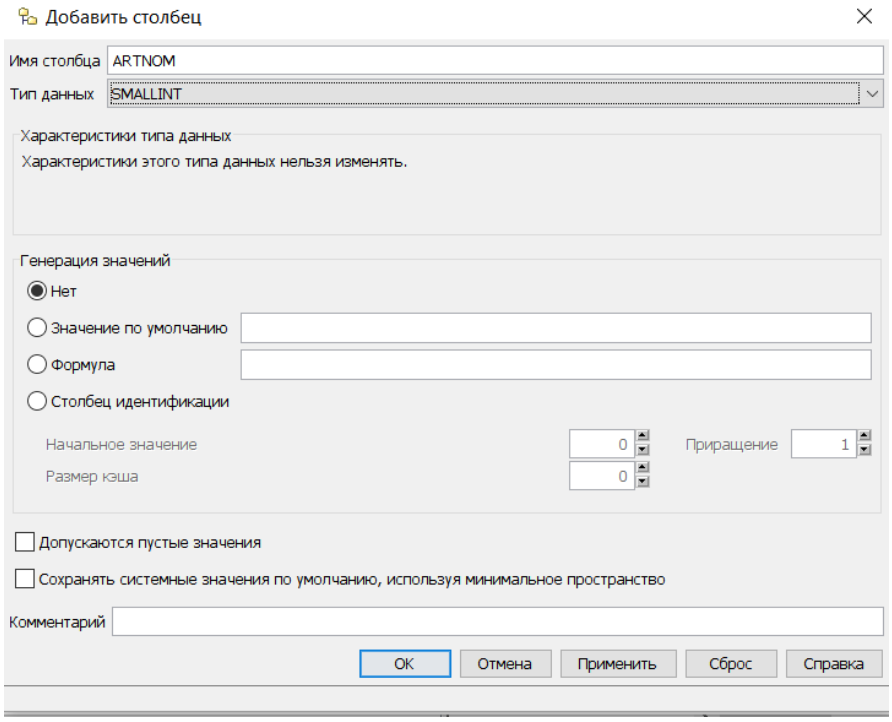


Рисунок 17.28 - Додавання атрибутів таблиці

Визначите стовпець *Name* (рис.17.29):

—Уведіть name у поле Ім'я стовпця (Column name).

—Виберіть VARCHAR зі списку, що розкривається, поля Тип даних (Data type).

—У поле Довжина (Length) уведіть значення 50.

—Відзначте прапорцем опцію Допускаються нульові значення (Nullable), щоб дозволити запис у цей стовпець дані зі значенням NULL.

—Натисніть Застосувати (Apply) потім ОК.

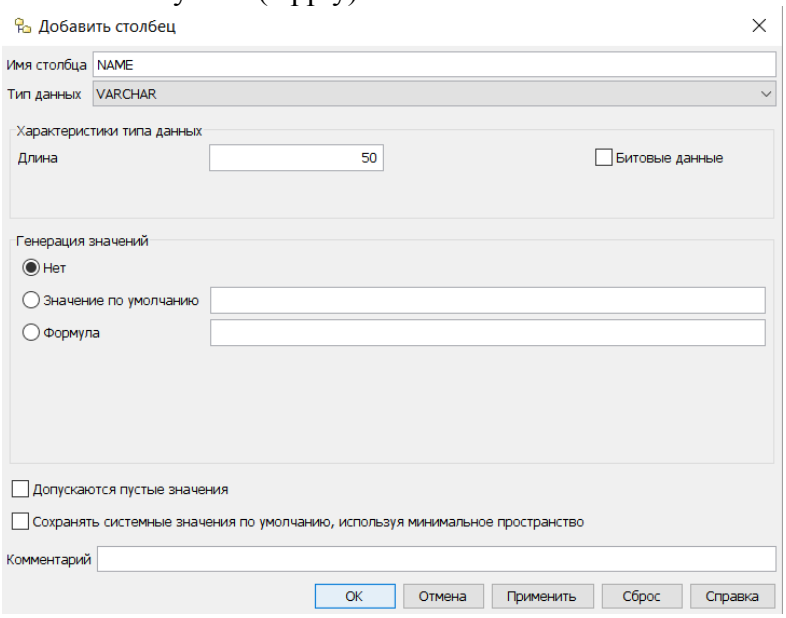


Рисунок 17.29 - Налаштування атрибуту *Name*

Визначите стовпець *classification*: уведіть classification у поле Ім'я стовпця (Column name). Виберіть CHARACTER зі списку, що розкривається, поля Тип даних (Data type).

У поле Довжина (Length) уведіть значення 1.

Зніміть прапорець із опції Допускаються нульові значення (Nullable), щоб заборонити запис у цей стовпець даних зі значенням NULL. Натисніть Застосувати (Apply) потім ОК.

Визначите стовпець bio (рис.17.30).

Уведіть bio у поле Ім'я стовпця (Column name).

Виберіть CLOB зі списку, що розкривається, поля Тип даних (Data type).

Уведіть значення 100 у поле Довжина (Length).

Виберіть Кбайт (Kbytes) зі списку, що розкривається, Одиниця більших об'єктів (LOB unit).

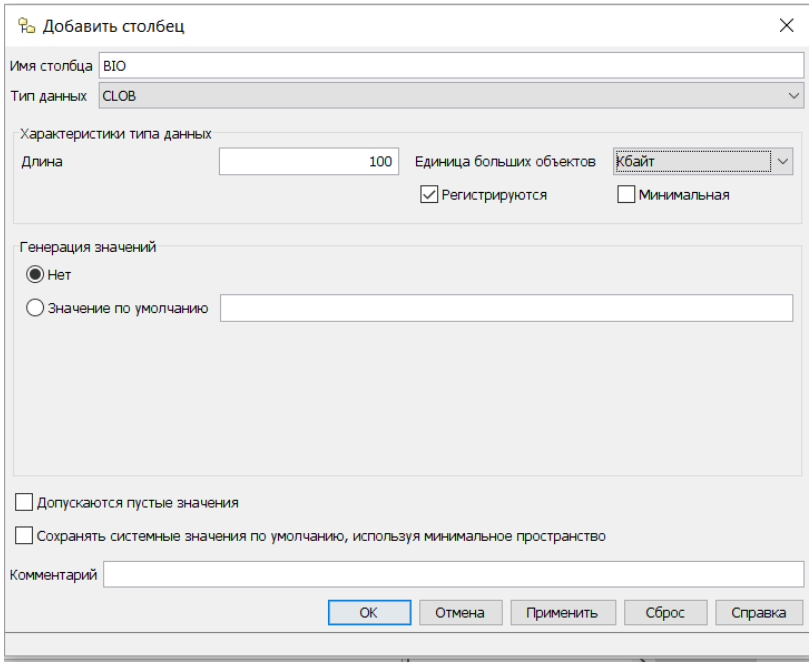


Рисунок 17.30 - Визначення стовпця BIO

У вікні «Задайте простір для зберігання таблиць» (рис.17.31), у випадку автоматичного створення табличного простору, натисніть кнопку «Далі».

Якщо була обрана « Настроювання користувача» - виберіть або створіть новий Табличний простір ля зберігання таблиць, індексів, CLOB, BLOB полів.

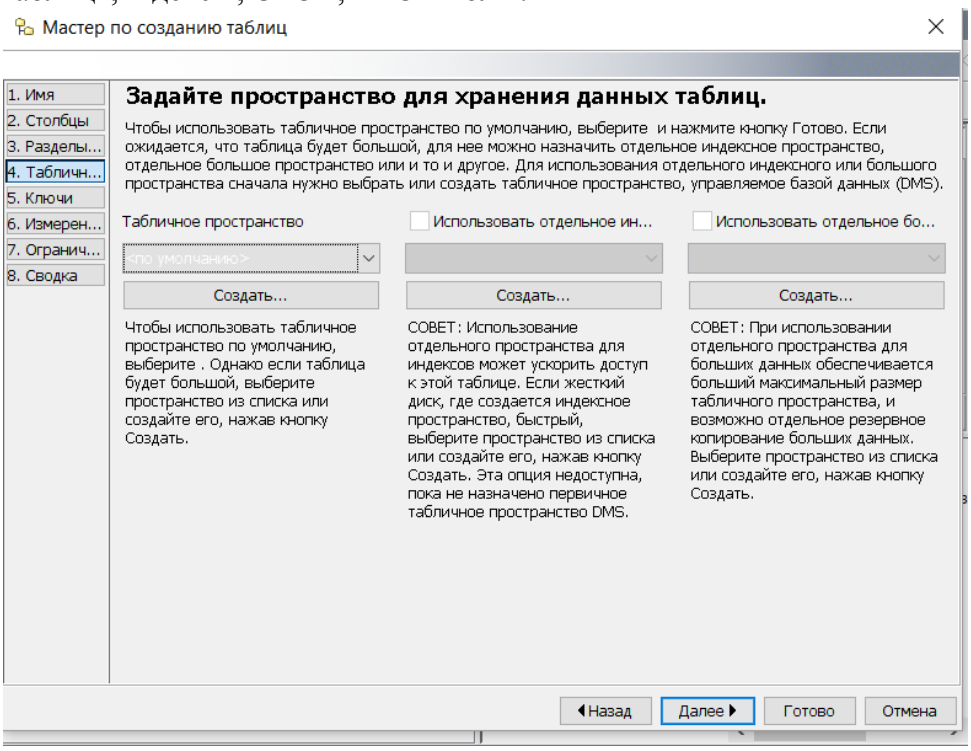


Рисунок 17.31 Вікно «Задайте простір для зберігання таблиць»

У вікні «Ключі» виберіть поле яке буде ключовим, додайте первинний ключ (рис.17.32).

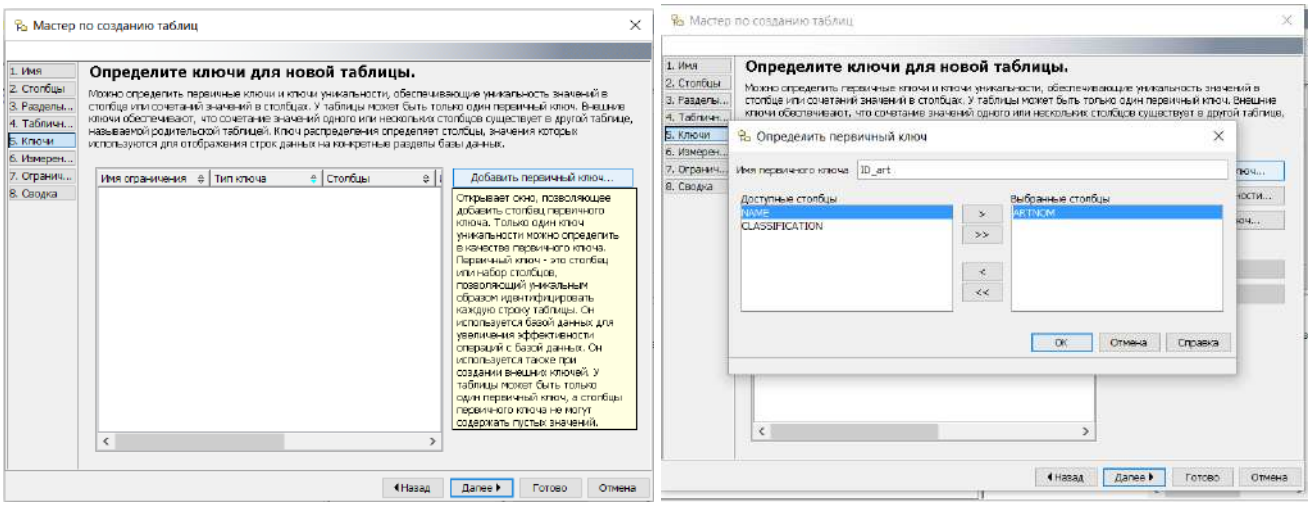


Рисунок 17.32 - Вікно «Визначення ключів таблиці»

Якщо таблиця дочірня, створіть Зовнішній ключ зв'язку, для чого виберіть батьківську таблицю й поле яке буде виконувати роль Зовнішнього ключа.

У випадку якщо планується зберігання полів таблиці на різних жорстких дисках у вікні «Задайте кластеризацию даних» (рис.17.33) укажіть шлях до папки.

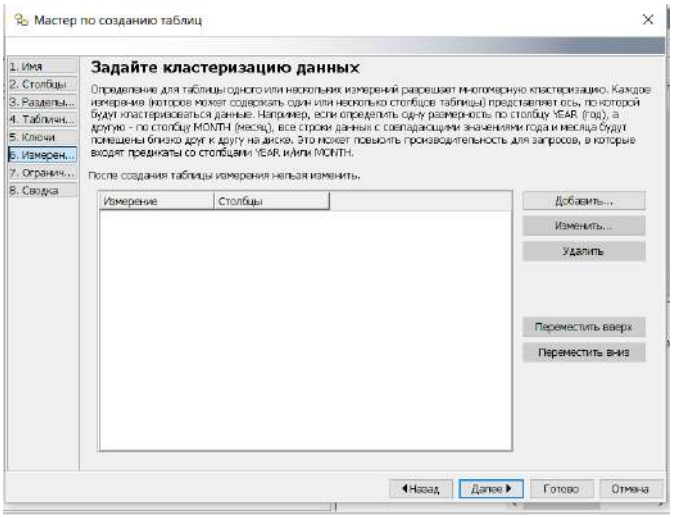


Рисунок 17.33- Вікно«Задайте кластеризацию даних»

Для забезпечення цілісності даних можна задати перевірочні обмеження для призначених полів (рис.17.34).

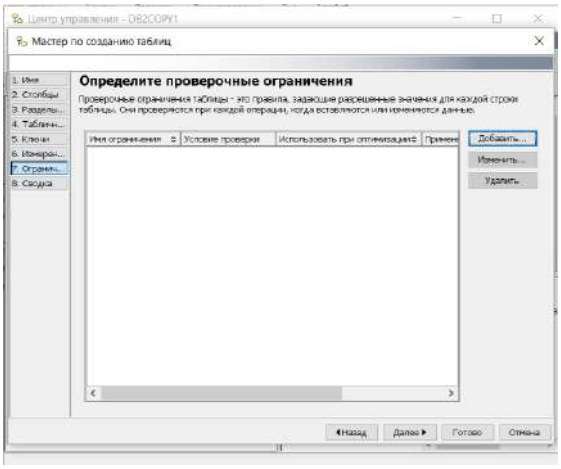


Рисунок 17.34 - Вікно для завдання перевірочних обмежень

У вікні «Відомості» можна перевірити обрані параметри таблиці. У випадку відсутності помилок видається повідомлення про успішне виконання операцій по створенню таблиці (рис.17.35).

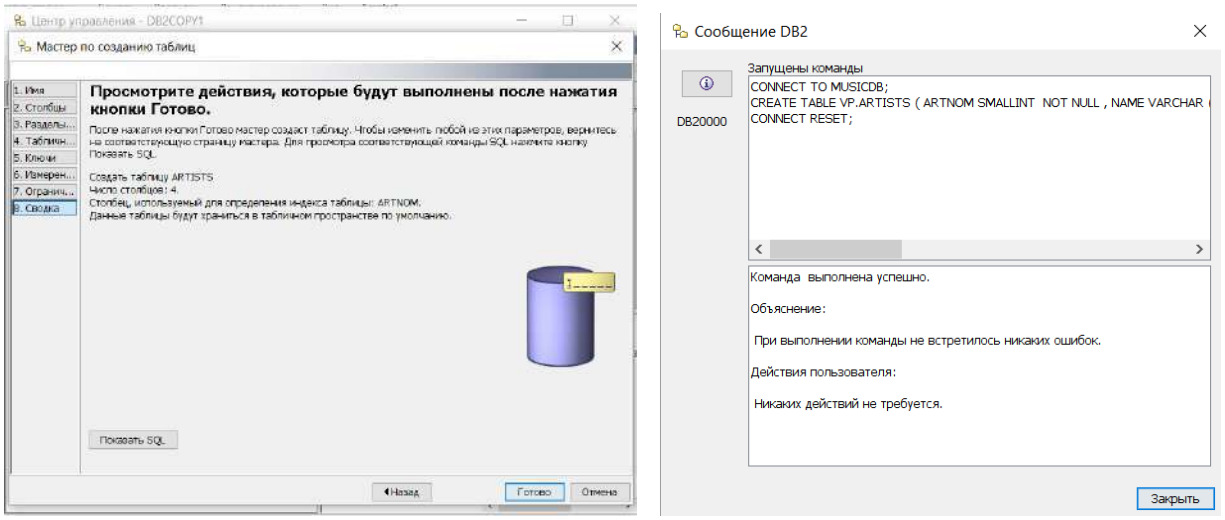


Рисунок 17.35 - Повідомлення про результат виконання операцій

Для уведення даних у таблицю відкрийте командою Відкрити майстер додавання записів (рис.17.36).

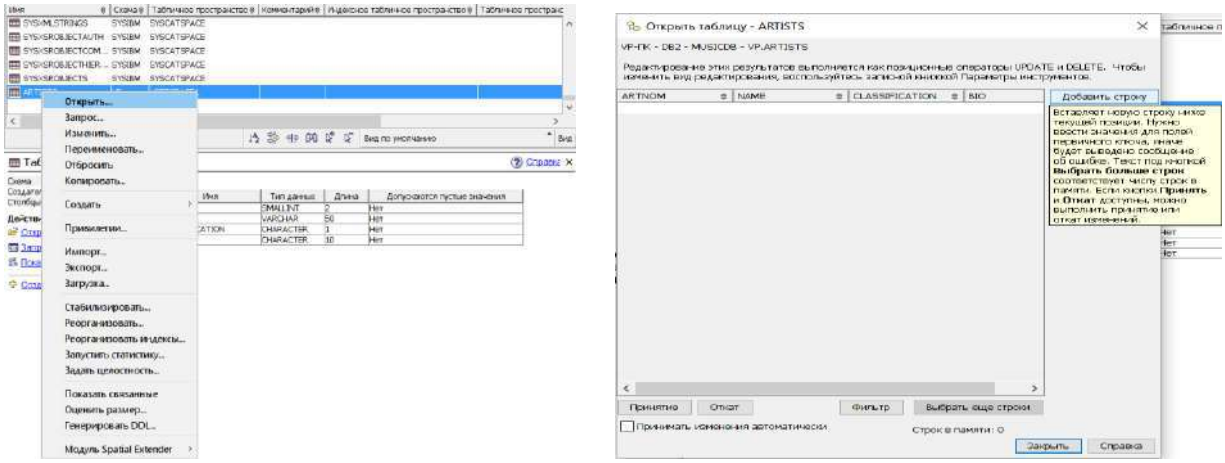


Рисунок 17.36 - Вікно для уведення даних у таблицю

Перевірте уведені дані, якщо буде потреба можна відредагувати запис (рис.17.37).

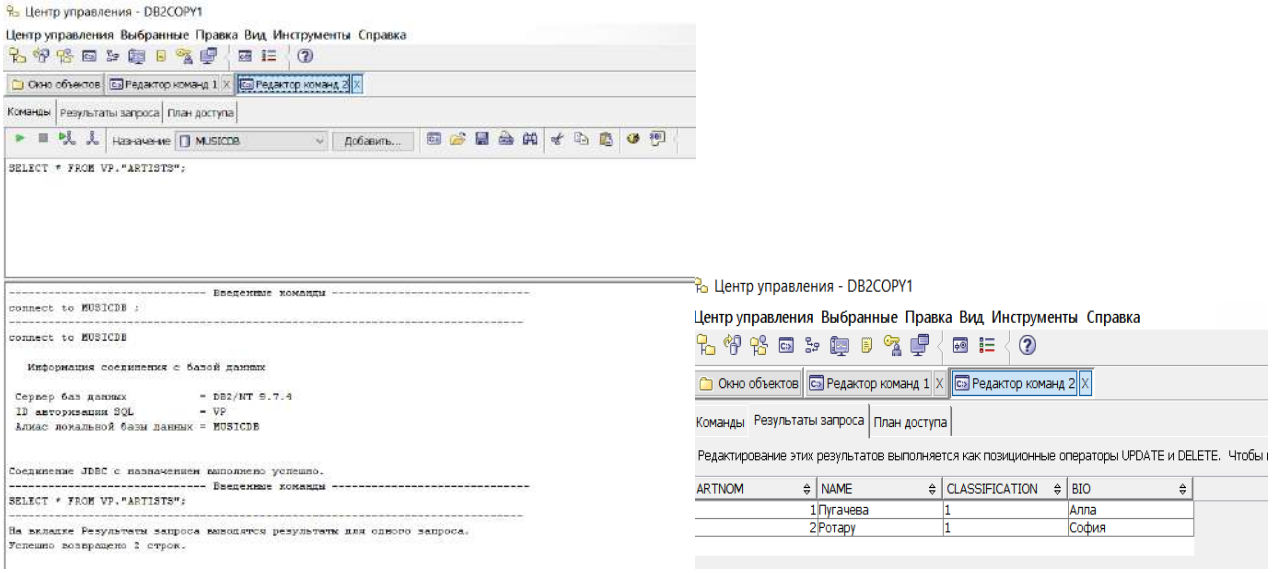


Рисунок 17.37 - Вікно для перевірки уведених даних у таблицю

Контрольні питання.

1. Визначення інформаційної системи і її можливості.
2. Призначення СКБД.
3. Основні архітектури побудови БД.
4. Які використовують засоби створення додатків для роботи з БД
5. Яке призначення має словник БД?
6. Поняття відношення, кортеж, домен. Надати визначення, навести приклади.
7. Поняття реляційної таблиці.
8. Первинні та зовнішні ключі відношення.
9. Які властивості характеризують реляційну модель даних ?
10. Призначення зв'язку між відношеннями.

18. Використання XML-даних у сервері IBM DB2

18.1 Основні визначення мови XML

Однією з найбільш істотних нових функціональних можливостей мови SQL: 2003 є підтримка файлів XML (extensible Markup Language - розширювана мова розмітки), які все більше стають універсальним стандартом обміну даними між різнорідними платформами. Для XML не має значення, у якому середовищі створені додатки, яка ОС або яке апаратне забезпечення працює у користувача, якому надаються дані. XML, як і HTML, є мовою розмітки, він ***не повнофункціональна*** мова, як C ++ або Java або мова маніпулювання даними, як SQL. Там, де HTML має справу тільки з форматуванням тексту і графіки в документі, XML дозволяє працювати зі структурою вмісту документа. Однак сам XML не призначений для прямого форматування. Для здійснення завдання форматування необхідно доповнити XML таблицею стилів, яка, як і в HTML, дозволяє форматувати XML-документи.

SQL і XML структурують дані двома різними способами. SQL представляє собою інструмент для роботи з числовими і текстовими даними, класифікованими за типами даних і мають точно описаний розмір. Мова SQL це стандартний інструмент для підтримки і роботи з даними, що зберігаються в реляційних базах даних. Якщо мова йде про довільних даних, які не можуть бути однозначно класифіковані, то тут краще використовувати мову XML.

Мета мови XML - забезпечити універсальний стандарт для переміщення даних між різнорідними комп'ютерами і відображення їх в WWW.

SQL і XML як би доповнюють цілі і можливості один одного. Кожен несе в собі певні неперевершені переваги щодо додаткив і форм, що дозволяє користувачеві отримати всю необхідну інформацію в будь-який час і в будь-якому місці.

Тип даних XML. У SQL: 2003 вперше був представлений новий для SQL тип даних - тип XML. Це означає, що узгоджені між собою реалізації можуть зберігати і працювати безпосередньо з даними формату XML, без їх попереднього перетворення з будь-якого типу даних SQL в тип XML. Незважаючи на те що XML-дані вбудовані в будь-яку підтримує їх реалізацію, вони працюють як тип даних, який визначається користувачем (UDT-тип, user defined type).

XML-дані забезпечують безпосередню взаємодію мов SQL та XML, оскільки дозволяють додаткам виконувати SQL-операції над вмістом XML і, навпаки, XML-операції над вмістом SQL. Ви можете використовувати стовпці з XML-даними спільно зі стовпцями, що містять будь-які зумовлені типи даних, об'єднуючи їх в запитах з пропозицією WHERE.

Коли використовуються XML-дані. Необхідність зберігання даних в форматі XML залежить від того, що ви плануєте робити з цим даними. Ухвалення такого рішення доцільно в наступних випадках:

- Якщо необхідно зберігати цілий блок даних для подальшого отримання цих даних в тому ж повному блоці.
- Якщо необхідно зробити запит для цілого XML-документа. Деякі реалізації мають розширені можливості оператора EXTRACT, які дозволяють отримати вміст з XML-документа.
- Якщо необхідно точно ввести дані поза операторів SQL. Використання типу даних XML гарантує істинність значень XML, а не тільки довільних текстових рядків.
- Для забезпечення сумісності з майбутніми, ще не встановленими системами зберігання, які можуть не підтримувати існуючі типи, наприклад CLOB.
- Для отримання переваг в майбутніх оптимізацію, які будуть підтримувати тільки XML-дані.

Іноді **використання XML-даних не має ніякого сенсу**, оскільки сьогодні більшість даних в реляційних базах даних набагато краще працює в їх поточному форматі, ніж в форматі XML.

XML тип не використовується в наступних випадках:

- При природному розбитті даних в реляційної структурі на таблиці, рядки і стовпці.
- Якщо необхідно оновити тільки деякі частини, а не весь документ.

18.2 Перетворення даних з SQL в формат XML

Перетворення наборів символів. Для обміну даними між базами даних SQL і XML-документами різні елементи бази даних SQL повинні бути перетворені в еквівалентні елементи XML-документа і навпаки.

У мові SQL підтримка наборів символів залежить від його реалізації. Це означає, що СУБД DB2 може підтримувати набори символів, які не підтримуються СУБД SQL Server (Microsoft). SQL Server, в свою чергу, підтримує набори символів, які не підтримуються додатком Oracle. Незважаючи на те що більшість загальних наборів символів практично завжди універсально підтримується тим або іншим СУБД, використання мало поширених символів може ускладнити переміщення бази даних і СУБД з однієї платформи реляційної СУБД на іншу.

У XML немає ніяких проблем сумісності з наборами символів, оскільки він підтримує тільки один набір - **Unicode.** Цей універсальний набір символів, що публікується консорціумом Unicode Consortium, являє собою стандартний набір символів для комп'ютерів, в якому кожному письмовою знаку на будь-якій мові ідентифікації абонента номер. Постачальники РСУБД повинні описати перетворення рядків з кожного набору символів в символи Unicode і зворотне перетворення символів Unicode в рядки, що складаються з наборів символів. На щастя, XML підтримує безліч наборів символів, що звільняє постачальників від великої кількості проблем, пов'язаних з незліченними перетвореннями символів.

Перетворення ідентифікаторів. На відміну від SQL, XML точніше підходить до визначення ідентифікаторів. Перш ніж стати частиною XML-документа, символи, допустимі в SQL і неприпустимі в XML, повинні бути відповідним чином перетворені.

SQL підтримує необмежені ідентифікатори. Це означає, що всі види додаткових символів, такі як %, $ і &, будуть допустимими до тих пір, поки вони укладені в подвійні лапки.

Але такі символи не допустимі для XML. Крім того, імена в XML, що починаються з символів XML, в будь-яких комбінаціях вже зарезервовані і, таким чином, не можуть використовуватися без будь-яких негативних наслідків. Саме тому ідентифікатори SQL, що починаються з цих символів, повинні бути змінені.

При перетворенні з SQL в XML все ідентифікатори конвертуються в Unicode. Будь-які ж ідентифікатори мови SQL, які є також допустимими іменами XML, залишаються незмінними. Символи ідентифікатора мови SQL, не припустимі для імен XML, замінюються шістнадцятковим кодом. Отриманий результат має форму записи типу "_xHHHH_" або "_xHHHHHHHH_", де H-шістнадцятковий розряд великих літер. Наприклад, символ підкреслення "_" буде представлений як "_x005F_". Двокрапка - як "_x003A_". Ці уявлення є кодами для опису в системі Unicode таких символів, як підкреслення і двокрапка. У разі, якщо ідентифікатор SQL починається з символів x, т або /, перед такими символами необхідно поставити префікс з кодом в формі "_xFFFF_".

Перетворити символи з XML-формату в SQL-формат набагато простіше. Все, що для цього необхідно зробити, - це розгорнути символи XML-імені в послідовність "XFFFF" або "XFFFFFFFF". Всякий раз, коли ви знаходите таку послідовність, замінюйте її символами, відповідними символам Unicode. Якщо ім'я XML починається з символів "XFFFF", ігноруйте їх.

Дотримуючись цих простих правил, ви можете перетворювати ідентифікатор SQL в XML-ім'я, а потім повернутися до ідентифікатора SQL. Однак це гарне правило не діє для перетворення XML-імені в ідентифікатор SQL і навпаки.

Перетворення типів даних. У SQL: 2003 визначено, що дані типу SQL перетворюються в найбільш близьку схему XML-даних. Формулювання "найбільш близька" означає, що всі значення, допустимі для SQL-типу, будуть допустимі і для типу XML-схеми, а найменш можливі значення, не припустимі для SQL-типу, будуть допустимі для типу XML-схеми. Елементи XML, такі як maxlnclusive і minlnclusive, можуть обмежувати значення, що допускаються типом XML-схеми, до значень, що допускаються відповідним SQL-типом.

Наприклад, якщо тип SQL-даних обмежує значення типу INTEGER в діапазоні значень від -2 157 483 648 до 2 157 483 647, в XML значення minlnclusive може бути задано числом -2157483648.

Приклад такого перетворення:

```
<xsd:simpleType>
<xsd:restriction base="xsd:integer">
<xsd:maxlnclusive value="2157483647"/>
<xsd:minlnclusive value="-21574 83648"/>
<xsd:annotation>
<sqlxml:sqltype name="INTEGER"/>
</xsd:annotation>
</xsd:restriction>
</xsd:simpleType>
```

Розділ приміток містить інформацію з опису SQL-типу, який в даний момент не використовується XML, але пізніше, при перетворенні цього документа в SQL-формат, може бути дуже до речі.

Перетворення таблиць. Ви можете перетворити таблицю в XML-документ, а також всі таблиці в схему або всі таблиці в каталог. Перетворення також визначає привілеї. Користувач, який має привілей SELECT тільки для декількох стовпців таблиці, може перетворити в XML-документ тільки ці стовпці. Насправді перетворення породжує два документа: один, який містить дані в таблиці, а інший - XML-схему, що описує перший документ.

Приклад перетворення SQL-таблиці в документ, що містить XML-дані.

```
<CUSTOMER>
<row>
<FirstName>Abe</FirstName>
<LastName>Abelson</LastName>
<City>Springfield</City>
<AreaCode>714</AreaCode>
<Telephone>555-llll</Telephone>
</row>
<row>
<FirstName>Bill</FirstName>
<LastName>Bailey</LastName>
<City>Decatur</City>
<AreaCode>714</AreaCode>
<Telephone>555-2222</Telephone>
</row>
</CUSTOMER>
```

Основний елемент документа дав ім'я таблиці (CUSTOMER). Кожен рядок таблиці міститься в межах елемента <row> (в даному прикладі таких рядків дві). Крім того, кожен рядок таблиці містить послідовність елементів стовпця (атрибутів), кожен з яких отримує ім'я після зв'язування зі стовпцем у вихідній таблиці (стовпчики FirstName, LastName, City, AreaCode, Telephone). Кожен елемент стовпця містить значення даних.

Обробка невизначених значень. Оскільки SQL-дані можуть бути невизначеними значеннями, необхідно вирішити, як представляти їх в XML-документі. Невизначені значення можуть бути представлені або як нуль, або як відсутність будь-якого значення. Якщо елемент стовпця необхідно представити як невизначений (нульове) значення, то він буде мати атрибут xsi: nil = "true".

Це може бути зроблено в такий спосіб:

```
<Row>
<FirstName> Bill </ FirstName>
<LastName> Bailey </ LastName>
<City xsi: nil = "true" />
<AreaCode> 714 </ AreaCode>
<Telephone> 5 55-22 22 </ Telephone>
</ Row>
```

Якщо елемент стовпця відсутня, то виконайте наступне:

<Row>
<FirstName> Bill </ FirstName>
<Las tName> Bailey </ Las tName>
<AreaCode> 714 </ AreaCode>
<Telephone> 55 5-22 22 </ Telephone>
</ Row>

При виборі цієї опції рядок містить невизначене значення, яке "відсутній". З ним немає зв'язків.

18.3 Створення XML-схеми

При перетворенні даних з SQL в XML перший створений документ містить дані, а другий - інформацію про схему.

Приклад схеми для документа CUSTOMER

```
<xsd:schema>
<xsd:simpleType name="CHAR_15">
<xsd:restriction base="xsd:string">
<xsd:lenght value = "15"/>
</xsd;restriction>
</xsd:simpleType>
<xsd:simpleType name="CHAR_25">
<xsd:restriction base="xsd:string">
<xsd:lenght value = "25"/>
</xsd:restriction>
</xsd:simpleType>
<xsd:simpleType name="CHAR_3">
<xsd:restriction base="xsd:string">
<xsd:lenght value = "3"/>
</xsd:restriction>
</xsd:simpleType>
<xsd:simpleType name-"CHAR_8">
<xsd:restriction base="xsd:string">
<xsd:lenght value = "8"/>
</xsd:restriction>
</xsd:simpleType>
<xsd:sequence>
<xsd:element name="FirstName" type="CHAR_15"/>
<xsd:element name="LastName" type="CHAR_25"/>
<xsd:element
name="City" type="CHAR_25" nillable="true"/>
<xsd:element
name="AreaCode" type="CHAR_3" nillable="true"/>
<xsd:element
name="Telephon" type="CHAR_8" nillable="true"/>
<xsd:sequence>
</xsd:schema>
```

Ця схема підходить в тому разі, якщо при обробці невизначених значень використовується опція "нуль". Опція "відсутності" вимагає дещо іншого визначення елемента.

Наприклад:

<Xsd: element

name = "City» type = "CHAR_25 minOccurs =" 0 "/>
Отримання XML результатів при використанні операторів SQL
SQL: 2003 існує п'ять операторів
XMLELEMENT,
XMLFOREST,
XMLGEN,
XMLCONCAT,
XMLAGG,
які при застосуванні до вмісту бази даних SQL дають XML-результат.

Оператор XMLELEMENT. Оператор XMLELEMENT створює XML-елемент. Цей оператор може використовуватися в операторі SELECT для переміщення даних формату XML в базу даних SQL.

Приклад:

```
SELECT c.LastName
XMLELEMENT (NAME "City", c.City) AS "Result"
FROM CUSTOMER З
WHERE LastName = "Abelson";
```

Повертається результат:

```
LastName Result
Abelson <City> Springfield </ City>
```

Оператор XMLFOREST. Оператор XMLFOREST створює дерево (forest) елементів зі списку аргументів. Кожен аргумент оператора створює новий елемент.

```
SELECT c. LastNartie
XMLFOREST (c.City,
c.AreaCode,
c.Telephone) AS "Result"
FROM CUSTOMER з
WHERE LastName = "Abelson" OR LastName = "Bailey";
```

В результаті отримаємо наступну інформацію:

```
LastName Result
Abelson <City> Springfield </ City>
<AreaCode> 714 </ AreaCode>
<Telephone> 555-llll </ Telephone>
Bailey <City> Decatur </ City>
<AreaCode> 714 </ AreaCode>
<Telephone> 555-llll </ Telephone>
```

Оператор XMLGEN. Перший аргумент оператора XMLGEN - це шаблон, який містить символи-заповнювачі для значень, які будуть додані пізніше. Символи-наповнювачі представлені в формі "{$ name}". Послідовність аргументів задає значення і пов'язані з ними імена, які характеризують шаблон.

```
SELECT c.LastName
XMLGEN ('<CUSTOMER Name="{$LASTNAME}">
<City>{$CITY}</City>
</CUSTOMERS,
c.LastName AS Name,
c City) AS "Result"
FROM CUSTOMER c
WHERE LastName="Abelson" OR LastName="Bailey";
```

Результат буде таким:

LastName	Result
Abelson	<CUSTOMER Name="Abelson" <City>Springfield</City> </CUSTOMER> <CUSTOMER Name="Bailey" <City>Decatur</City> </CUSTOMER>

Оператор XMLCONCAT забезпечує альтернативний спосіб створення дерева елементів. Це здійснюється шляхом зв'язування його XML-аргументів.

наприклад:

```
SELECT c.LastName
XMLCONCAT
(
XMLELEMENT (NAME "first", c.FirstName,
XMLELEMENT (NAME "last", c.LastName)
) AS "Result"
FROM CUSTOMER c;
```

В результаті вийде наступне:

LastName Result
Abelson <first> Abe </ first>
<Last> Abelson </ last>
Bailey <first> Bill </ first>
<Last> Bailey </ last>

Оператор XMLAGG. XMLAGG - це функція, яка створює єдиний XML-документ з інших XML-документів або їх окремих фрагментів. Агрегування містить дерево елементів. Розглянемо це на прикладі:

```
SELECT XMLELEMENT
(NAME "City",
XMLATTRIBUTES (c.City AS "name"),
XMLLAGG (XMLELEMENT (NAME "last" c.LastNa, e)
)
) AS "CityList"
FROM CUSTOMER З
GROUP BY City;
```

При обробці таблиці CUSTOMER цей запит виведе наступне:

<City name = "Decatur">
<Last> Bailey </ last>
</ City>
<City name = "Philo">
<Last> Stetson </ last>
<Last> Stetson </ last>
<Last> Wood </ last>
</ City
<City name = "Springfield">
<Last> Abelson </ last>

18.4 Створення таблиць які містять тип даних XML

Якщо при створені бази даних вибрано «за замовчуванням автоматичне сховище» включено це може підвищити продуктивність і керування даними XML, оскільки воно дає простір таблиць, кероване базами даних (DMS), що розширюється в міру необхідності. Крім того, за замовчуванням у базах даних використається набір кодів **UTF-8 (Unicode).**

Для створення бази даних для зберігання XML потрібно виконати команду:

create database* xmltut *using codeset UTF-8 territory us

Якщо ви вирішили зберігати XML-дані в базі даних з кодом, відмінним від UTF-8, краще вставити ці дані у форму, що не перетерплює перетворення кодової сторінки, наприклад, BIT DATA, BLOB або XML. Щоб заблокувати використання типів символьних даних під час синтаксичного аналізу XML, запобігаючи можливій заміні символів, установите для параметра конфігурації **ENABLE_XMLCHAR** значення **No**.

Для підключення до БД виконайте команду:

CONNECT TO xmltut;

Приклад створення таблиці:

```
CREATE TABLE VP.Customer
(Cid SMALLINT NOT NULL PRIMARY KEY,
 Info XML);
```

Первинний ключ не потрібно для зберігання або індексування даних XML. Можна додати один або кілька стовпців XML у таблицю, використовуючи інструкцію **ALTER TABLE.**

Створення індексів по XML-даним. Індекси над XML-даними можуть підвищити продуктивність запитів у стовпцях XML. У реляційному індексі і індексі по XML-даним індексуються стовпці. Однак реляційній індекс індексує весь стовпець, а індекс по даним XML індексує частина стовпця. Для цього потрібно вказати, які частини стовпця XML індексуються, указавши шаблон XML, що є обмеженим вираженням XPath. Також потрібно вказати тип даних, у якому будуть зберігатися індексовані значення. Як правило, обраний тип повинен бути того ж типу, що використається в запитах. Як і реляційні індекси, рекомендується індексувати XML-елементи або атрибути, які часто використаються в предикатах і об'єднаннях між документами. **Можна індексувати тільки один стовпець XML**. Складені індекси не підтримуються, можна мати кілька індексів у стовпці XML. Не всі пропозиції оператора CREATE INDEX застосовуються до індексів по XML-даним.

Щоб створити індекс над даними XML, виконаєте наступну команду (рис.18.1):

```
CONNECT TO xmltut;
CREATE INDEX cust_cid_xmlidx ON VP. Customer(Info)
GENERATE KEY USING XMLPATTERN
 'declare default element namespace "http://posample.org"; /customerinfo/@Cid'
AS SQL DOUBLE;
```

Оператор індексує значення атрибута ***cid*** елементів <customerinfo> зі стовпця INFO таблиці CUSTOMER.

Як значення за замовчуванням, коли дані XML індексуються, якщо дані XML не можуть бути передані зазначеному типу даних, SQL DOUBLE, індексний запис не створюється, і ніяка помилка не повертається.

Шаблон XML чутливий до регістра. Якщо XML-документи містили атрибут ***cid*** замість ***Cid***, ці документи не відповідали б цьому індексу

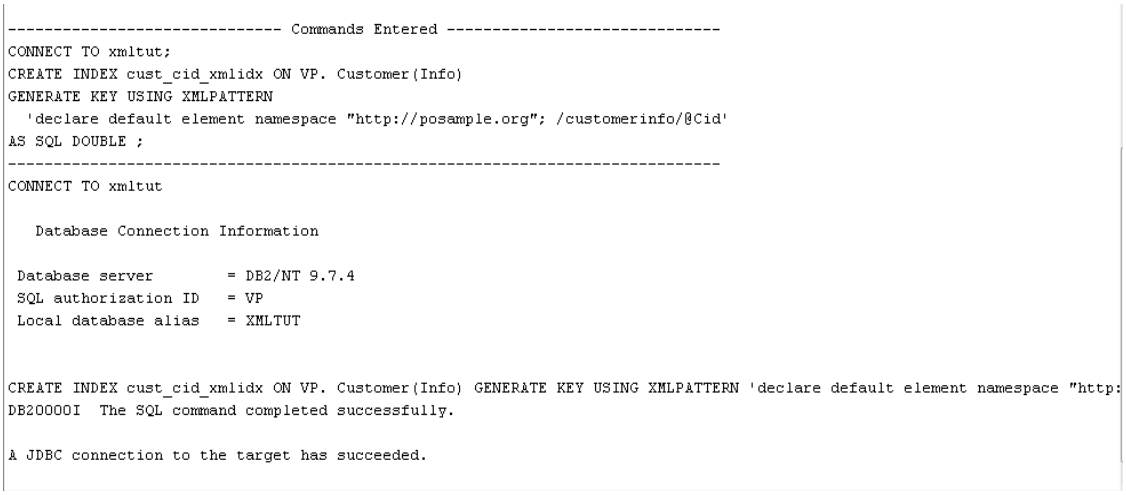


Рисунок 18.1 - Лістинг коду для створення індексу

18.5 Вставка XML-документів у стовпці типу XML

Добре сформовані XML-документи уставляються в стовпці з типом XML, використовуючи інструкцію **INSERT.** Можна вручну вставляти XML-документи в стовпці типу XML, використовуючи процесор командного рядка або за допомогою прикладних програм.

У більшості випадків не можна прямо призначати строкові дані цільовому типу з типом даних XML, спочатку потрібно проаналізувати дані за допомогою функції **XMLPARSE**.

В операціях **INSERT, UPDATE** або **DELET** можна призначати строкові дані безпосередньо в стовпці XML без явного виклику функції XMLPARSE. У цих трьох випадках строкові дані неявно аналізуються.

Щоб вставити три XML-документи в таблицю, наприклад, VP. Customer (VP- ім'я схеми, збігається з ім'ям користувача ПК) , виконаєте наступні інструкції:

CONNECT TO xmltut;

```
INSERT INTO Customer (Cid, Info)
 VALUES (2,
'<customerinfo xmlns="http://posample.org" Cid="1000">
  <name>Kathy Smith</name>
  <addr country="Canada">
    <street>5 Rosewood</street>
    <city>Toronto</city>
    <prov-state>Ontario</prov-state>
    <pcode-zip>M6W 1E6</pcode-zip>
  </addr>
  <phone type="work">416-555-1358</phone>
</customerinfo>');
```

При відсутності синтаксичних помилок відкриється повідомлення наступного виду (рис.18.8).

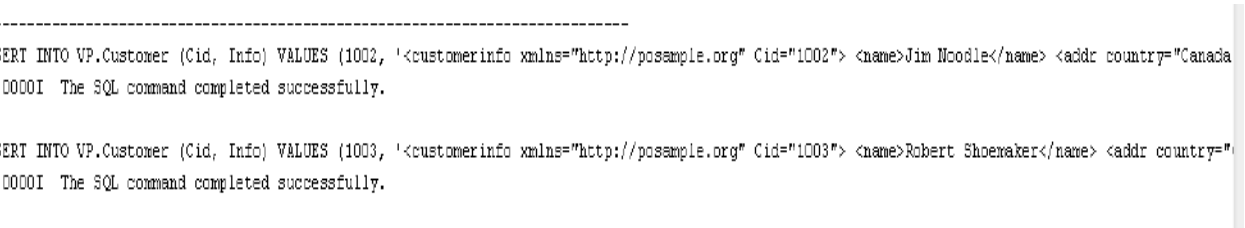


Рисунок 18.2 - Лістинг коду для виконання команди INSERT

Якщо допущені помилки, варто прочитати повідомлення й знайти в таблиці по коду помилки її зміст (рис.18.3):

```
112     '/' not allowed after '//' in XPath
113     '//' only allowed after '.' At the beginning of an XPath
114     Not allowed to have '/' at the beginning of an XPath value
115     Not allowed to select the root of an XPath
116     Empty XPath expression
117     The XPath expression cannot end with '|'
118     Invalid character following '.' in XPath
119     XPath token not supported
120     Enabling the NEL options
121     No scheme was found in the URI
122     The derived complexType has content while the base type is
        empty
123     The parser expected to find an XML notation name and did not.
124     An unterminated notation declaration was found
125     The parser did not find a document encoding but one was
        required.
126     The document contained more end tags than start tags
```

Рисунок 18.3 - Лістинг коду помилок

Щоб перевірити, що записи були успішно вставлені (рис.18.4), уведіть наступну інструкцію:

```
SELECT * from Customer
```

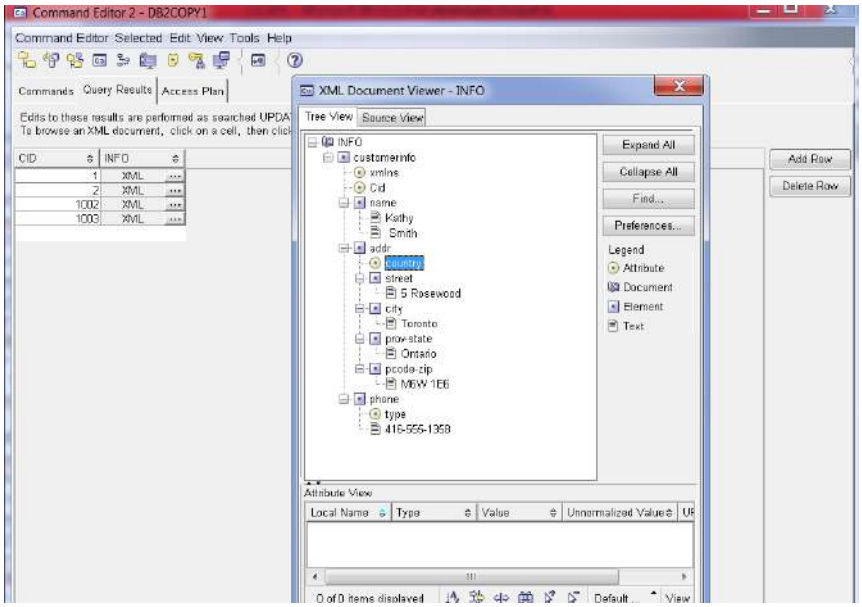


Рисунок 18.4 - Перегляд доданеої записі

18.6 Відновлення XML-документів, що зберігаються в стовпцях XML

Обновити XML-документи можна використовуючи інструкцію UPDATE SQL з вираженням відновлення XQuery або без нього.

Відновлення без вираження відновлення XQuery. Якщо використовувати інструкцію UPDATE без вираження відновлення XQuery, потрібно виконати відновлення повного докуме-

нта. Щоб обновити значення елементів <street>, <city> і <pcode-zip> для одного з документів виконаєте наступну команду:

```
INSERT INTO VP.Customer (Cid, Info) VALUES (1002,
'<customerinfo xmlns="http://posample.org" Cid="1002">
  <name>Jim Noodle</name>
  <addr country="Canada">
    <street>25 EastCreek</street>
    <city>Markham</city>
    <prov-state>Ontario</prov-state>
    <pcode-zip>N9C 3T6</pcode-zip>
  </addr>
  <phone type="work">905-555-7258</phone>
</customerinfo>');

UPDATE VP.customer SET info =
'<customerinfo xmlns="http://posample.org" Cid="1002">
  <name>Jim Noodle</name>
  <addr country="Canada">
    <street>1150 Maple Drive</street>
    <city>Newtown</city>
    <prov-state>Ontario</prov-state>
    <pcode-zip>Z9Z 2P2</pcode-zip>
  </addr>
  <phone type="work">905-555-7258</phone>
</customerinfo>'
WHERE XMLEXISTS (
 'declare default element namespace "http://posample.org";
  $doc/customerinfo[@Cid = 1002]'
  passing INFO as "doc")
```

Предикат **XMLEXISTS** гарантує, що заміняється тільки документ, що містить атрибут Cid = «1002». Предикатне вираження в XMLEXISTS, [@Cid = 1002], не зазначено у вигляді порівняння рядків: [@Cid = "1002"]. Причина в тім, що використали тип даних DOUBLE, коли створили індекс для атрибута Cid. Щоб індекс відповідав цьому запиту, ви не можете вказати Cid як рядок у вираженні предиката.

Щоб переверіть, що документ XML був обновлений, уведіть наступну інструкцію:

```
SELECT * from Customer
```

Запис, що містить Cid = «1002», містить змінені значення <street>, <city> і <pcode-zip>.

Якщо можна ідентифікувати XML-документи в таблиці, використовуючи значення в стовпцях не-XML однієї таблиці, використовують предикати порівняння SQL, щоб визначити, які рядки обновляти.

У попередньому прикладі, коли значення Cid з документа XML також зберігається в стовпці CID таблиці CUSTOMER, ви могли б використати предикат порівняння SQL у стовпці CID для ідентифікації рядка. У попередньому прикладі ви можете замінити пропозицію WHERE наступною пропозицією: `WHERE Cid=1002`

```
CONNECT TO xmltut;

UPDATE VP.customer SET info =
'<customerinfo xmlns="http://posample.org" Cid="1002">
  <name>Jim Noodle</name>
```

```
  <addr country="Canada">
    <street>1150 Maple Drive</street>
    <city>Newtown</city>
    <prov-state>Ontario</prov-state>
    <pcode-zip>Z9Z 2P2</pcode-zip>
  </addr>
  <phone type="work">905-555-7258</phone>
</customerinfo>'
WHERE Cid=1002;
```

```
CONNECT TO xmltut

   Database Connection Information

 Database server        = DB2/NT 9.7.4
 SQL authorization ID   = VP
 Local database alias   = XMLTUT


UPDATE VP.customer SET info = '<customerinfo xmlns="http://posample.org" Cid="1002"> <name>Jim Noodle</name> <addr country="Canada"> <street>1150 Maple Driv
DB20000I  The SQL command completed successfully.
```

Відновлення за допомогою вираження відновлення XQuery. Якщо використовувати інструкцію UPDATE з вираженням відновлення XQuery, **можна обновляти частини існуючого** XML-документа.

Щоб обновити адреса клієнта в існуючому XML-документі, уведіть наступний оператор SQL, у якому використається вираження перетворення XQuery:

```
UPDATE Customer set Info =
  XMLQUERY( 'declare default element namespace "http://posample.org";
  transform
  copy $mycust := $cust
  modify
    do replace $mycust/customerinfo/addr with
      <addr country="Canada">
        <street>25 EastCreek</street>
        <city>Markham</city>
        <prov-state>Ontario</prov-state>
        <pcode-zip>N9C 3T6</pcode-zip>
     </addr>
  return $mycust'
  passing INFO as "cust")
WHERE CID = 1002~
```

Щоб обновити адреса клієнта, функція XMLQUERY виконує вираження перетворення XQuery, що використає вираження replace, а потім повертає обновлену інформацію в оператор UPDATE у такий спосіб:

- Пропозиція передачі XMLQUERY використає обмеження ідентифікатора для передачі інформації про клієнта у вираження XQuery зі стовпця XML INFO.
- У пропозиції copy вираження transform приймається логічний знімок інформації про клієнта й привласнюється змінної $ mycust.
- У пропозиції зміни вираження transform вираження replace заміняє адресну інформацію в копії інформації про клієнта.
- XMLQUERY повертає обновлений клієнтський документ у змінної $ mycust.

Щоб підтвердити, що XML-документ містить обновлену адресу клієнта, виконаєте наступну команду: `SELECT Info FROM Customer WHERE Cid = 1002`

18.7 Видалення даних XML

Видалення цілих XML-документів. Щоб видалити цілі XML-документи, використайте оператор **DELETE** SQL. Використайте предикат **XMLEXISTS** для визначення окремих документів для видалення.

Щоб видалити тільки XML-документи з елементом <customerinfo> з атрибутом Cid = 1003 зі стовпця Info, виконаєте наступну команду:

```
DELETE FROM Customer
WHERE XMLEXISTS (
 'declare default element namespace "http://posample.org";
  $doc/customerinfo[@Cid = 1003]'
  passing INFO as "doc")
```

Якщо можна ідентифікувати XML-документи в таблиці, використовуючи значення в стовпцях не-XML однієї таблиці, можна використати предикати порівняння SQL, щоб визначити, які рядки видалити. У попередньому прикладі, коли значення Cid з XML-документа також зберігається в стовпці CID таблиці CUSTOMER, можна б виконати ту ж операцію, використовуючи наступний оператор DELETE, що застосовує предикат порівняння SQL зі стовпцем CID для визначити рядок:

```
DELETE FROM Customer WHERE Cid=1003
```

Щоб підтвердити, що документи XML були вилучені, виконаєте наступну інструкцію :

```
SELECT * FROM Customer
```

Повертаються два записи.

Видалення розділів XML-документів. Щоб видалити тільки розділи XML-документа замість видалення всього документа, використайте оператор UPDATE SQL, що містить вираження для видалення XQuery.

Щоб видалити всю телефонну інформацію із запису клієнта, де значення Cid дорівнює 1002, уведіть наступний оператор SQL, що використає функцію XMLQUERY:

```
UPDATE Customer
SET info = XMLQUERY(
   'declare default element namespace "http://posample.org";
    transform
    copy $newinfo := $info
    modify do delete ($newinfo/customerinfo/phone)
    return $newinfo' passing info as "info")
WHERE cid = 1002
```

Щоб видалити елемент <phone>, функція XMLQUERY виконує вираження перетворення XQuery, що використає вираження delete, а потім повертає обновлену інформацію в оператор UPDATE у такий спосіб:

- Пропозиція передачі XMLQUERY використає інформацію ідентифікатора для передачі інформації про клієнта у вираження XQuery зі стовпця XML INFO.
- У пропозиції копіювання вираження перетворення береться логічний знімок інформації про клієнта й привласнюється змінної $ newinfo.
- У пропозиції зміни вираження перетворення вираження delete видаляє елемент <phone> у копії інформації про клієнта.

XMLQUERY повертає обновлений клієнтський документ у змінної $ newinfo.

Щоб підтвердити, що запис клієнта більше не містить елемент <phone>, уведіть наступну інструкцію:

```
SELECT * FROM Customer WHERE Cid=1002
```

18.8 Створення запиту до даних XML

Якщо використається тільки SQL, можна запитувати тільки рівень стовпця, тобто можна повернути весь XML-документ, що зберігається в стовпці, *але* **не можна запитувати усередині документа** або повертати **фрагменти документа**. Чтобы запросити значення в документі XML або повернути фрагменти документа, повинні використати XQuery. Запити використають XQuery у контексті SQL і SQL у контексті XQuery.

XQuery чутливий до регістра. Тому при використанні XQuery ретельно вказуйте імена, такі як імена таблиць і SQL-схем, які за замовчуванням є заголовними. Навіть у контексті SQL вираження XQuery залишаються чутливими до регістра.

Запит у контексті SQL. Одержання цілих XML-документів.

Щоб одержати всі XML-документи, що зберігаються в стовпці INFO і значення зі стовпця первинного ключа CID, виконаєте наступну інструкцію SELECT:

```
SELECT cid, info FROM customer
```

Цей запит повертає два збережених XML-документи.

Витяг і фільтрація значень XML. Щоб запросити документи XML у стовпці INFO, виконаєте наступну інструкцію SELECT, що використає функцію XMLQUERY для виклику вираження XQuery:

```
SELECT XMLQUERY (
  'declare default element namespace "http://posample.org";
   for $d in $doc/customerinfo
   return <out>{$d/name}</out>'
   passing INFO as "doc")
FROM Customer as c
WHERE XMLEXISTS ('declare default element namespace "http://posample.org";
   $i/customerinfo/addr[city="Toronto"]' passing c.INFO as "i")
```

У функції XMLQUERY спочатку вказується простір імен за замовчуванням. Це простір імен збігається із простором імен раніше вставлених документів.

Пропозиція *for* указує ітерацію через елементи <customerinfo> у кожному документі зі стовпця Info. Стовпець INFO указується за допомогою пропозиції передачі, що зв'язує стовпець INFO зі змінної doc, на яку посилається в пропозиції for.

Пропозиція *return* потім створює елемент <out>, що містить елемент <name> від кожної ітерації пропозиції for.

Пропозиція WHERE використає предикат XMLEXISTS, щоб розглядати тільки підмножину документів у стовпці «Інформація». Ця фільтрація дає тільки ті документи, у яких є елемент <city> (по зазначеному шляху) зі значенням Торонто.

Оператор SELECT повертає наступний побудований елемент:

```
<out xmlns="http://posample.org"><name>Kathy Smith</name></out>
```

Використання db2-fn: sqlquery з параметрами.Щоб передати значення SQL fullselect у функції db2-fn: sqlquery, запустите наступний запит:

```
VALUES XMLQUERY (
  'declare default element namespace "http://posample.org";
  for $d in db2-fn:sqlquery(
    ''SELECT INFO FROM CUSTOMER WHERE Cid = parameter(1)'',
```

```
    $testval)/customerinfo
  return <out>{$d/name}</out>'
  passing 1000 as "testval" )
```

Функція XMLQUERY передає значення 1000 у вираження XQuery з використанням ідентифікатора testval. Потім вираження XQuery передає значення функції db2-fn: sqlquery за допомогою скалярної функції PARAMETER.

Вираження XQuery повертає наступний побудований елемент:

```
<out xmlns="http://posample.org">
   <name>Kathy Smith</name>
</out>
```

Запит у контексті XQuery. DB2® XQuery пропонує дві убудовані функції, спеціально призначені для використання з базами даних DB2: db2-fn: ***sqlquery*** і db2-fn: ***xmlcolumn***.

db2-fn: sqlquery витягає послідовність, що є таблицею результатів SQL fullselect.

db2-fn: xmlcolumn витягає послідовність зі стовпця XML.

Якщо запит викликає вираження XQuery прямо, встановити префікс його за допомогою ключового слова XQUERY без обліку регістра. Існує кілька параметрів, які можна настроїти для настроювання середовища процесора командного рядка, особливо для відображення результатів вираження XQuery.

Наприклад, установите параметр -i, щоб спростити читання результатів з виражень XQuery у такий спосіб: `UPDATE COMMAND OPTIONS USING i ON`

Одержання цілих XML-документів. Щоб одержати всі XML-документи, раніше вставлені в стовпець INFO, можна використати XQuery з **db2-fn: xmlcolumn** або **db2-fn: sqlquery**.

Використання db2-fn: xmlcolumn. Щоб одержати всі XML-документи в стовпці INFO, виконаєте наступний запит:

```
XQUERY db2-fn: xmlcolumn ('CUSTOMER.INFO')
```

Імена операторів SQL автоматично перетворяться у верхній регістр за замовчуванням. Тому, коли створили таблицю CUSTOMER за допомогою інструкції CREATE TABLE SQL, імена таблиці і стовпців були зроблені заголовними. Оскільки XQuery чутливий до регістра, потрібно використати правильний випадок при вказівці імен таблиць і стовпців при використанні db2-fn: xmlcolumn. Цей запит еквівалентний SQL-запиту `SELECT Info FROM Customer.`

Використання db2-fn: sqlquery. Щоб одержати всі XML-документи в стовпці INFO, виконаєте наступний запит:

```
XQUERY db2-fn: sqlquery ('SELECT Info FROM Customer')
```

Не потрібно вказувати імена INFO і CUSTOMER у верхньому регістрі, тому що оператор SELECT обробляється в контексті SQL і тому не чутливий до регістра.

Одержання часткових документів XML. Щоб витягати фрагменти документа і фільтрувати значення, що є присутнім у документі, використовують XQuery за допомогою або db2-fn: xmlcolumn, або db2-fn: sqlquery.

Використання db2-fn: xmlcolumn. Щоб повернути елементи, що містять <name> вузли для всіх документів у стовпці Info, які мають елемент <city> (по зазначеному шляху) зі значенням Toronto, виконаєте наступний запит:

```
XQUERY declare default element namespace "http://posample.org";
  for $d in db2-fn:xmlcolumn('CUSTOMER.INFO')/customerinfo
  where $d/addr/city="Toronto"
  return <out>{$d/name}</out>
```

Функція db2-fn: xmlcolumn витягає послідовність зі стовпця INFO таблиці CUSTOMER.

Пропозиція for прив'язує змінну $ d до кожного елемента <customerinfo> у стовпці CUSTOMER.INFO. Пропозиція where обмежує елементи тільки тими, у яких є елемент <city> (по зазначеному шляху) зі значенням Торонто.

Пропозиція return повертає повертає значение, ЩО, XML.

Це значення є елементом <out>, що містить елемент <name> для всіх документів, які задовольняють умові, зазначеному в пропозиції where, у такий спосіб:

```
<out xmlns="http://posample.org">
<name>
              Kathy Smith
</name>
</out>
```

Використання db2-fn: sqlquery. Щоб задати повний вибір у вираженні XQuery, виконаєте наступний запит:

```
XQUERY declare default element namespace "http://posample.org";
  for $d in db2-fn:sqlquery(
    'SELECT INFO
     FROM CUSTOMER
     WHERE Cid < 2000')/customerinfo
  where $d/addr/city="Toronto"
  return <out>{$d/name}</out>
```

У цьому прикладі набір запитуваних XML-документів спочатку обмежується в повному виді конкретними значеннями в стовпці CID не XML. Цей приклад демонструє перевага db2-fn: sqlquery: ***він дозволяє застосовувати предикати SQL у вираженні XQuery***.

Потім документи, які є результатом SQL-запиту, далі обмежені в пропозиції where вираження XQuery тими документами, у яких є елемент <city> (по зазначеному шляху) зі значенням Toronto. Запит дає ті ж результати, що й у попередньому прикладі, що використав db2-fn: xmlcolumn:

```
<out xmlns="http://posample.org">
<name>
              Kathy Smith
</name>
</out>
```

Використання db2-fn: sqlquery з параметрами. Щоб передати значення SQL fullselect у функції db2-fn: sqlquery, запустите наступний запит:

```
XQUERY declare default element namespace "http://posample.org";
  let $testval := 1000
  for $d in db2-fn:sqlquery(
    'SELECT INFO FROM CUSTOMER WHERE Cid = parameter(1)',
    $testval)/customerinfo
  return <out>{$d/name}</out>
```

У вираженні XQuery пропозиція let установлює значення $ testval рівним 1000. У пропозиції for вираження потім передає значення функції db2-fn: sqlquery з використанням скалярної функції PARAMETER.

Вираження XQuery повертає наступний побудований елемент**:**

```
<out xmlns="http://posample.org">
   <name>Kathy Smith</name>
</out>
```

19. Моделі та концепції проектування баз знань

19.1 Базові поняття

Знання - це виявлені закономірності, принципи, зв'язки, закони у заданої предметної області, що дозволяють вирішувати інформаційні та виробничі завдання, приймати рішення. Знання пов'язані з даними, ґрунтуються на них, але представляють результат розумової діяльності людини, узагальнюють його досвід, отриманий в ході виконання якої-небудь практичної діяльності. Вони є результатом емпіричного досвіду [15,16].

При обробці на ЕОМ знання трансформуються аналогічно даним:

- знання в пам'яті людини як результат мислення;
- матеріальні носії знань (підручники, методичні посібники);
- поле знань - умовний опис основних об'єктів наочної області, їх атрибутів і закономірностей, що їх зв'язують;
- знання, описані на мовах подання знань (продукційні мови, семантичні мережі, фрейми);
- бази знань.

Часто використовуються такі визначення знань: **знання - це добре структуровані дані, або дані про дані, або метадані.**

Інформаційна система може видавати користувачу інформацію, підказку або рішення але не всі вони є знаннями. Існують такі розмежування в поняттях інформації, даних та знань:

- дані – не оброблені факти якої либо предметної області;
- інформація – структурований набір даних предметної області;
- знання – інформація, що піддається інтерпретації (*meaningful information*) та має чітку мету та вирішує інформаційні завдання у предметної області.

Інформацію, що зберігається у базі знань, можна поділити на зовнішню, внутрішню і корпоративну

До зовнішньою інформації і даних можна віднести:

- друкарські видання, бібліотечна інформація;
- звіти про відвідини виставок, конференцій і семінарів;
- звіти про переговори із замовниками;
- результати зовнішніх аудитів;
- зовнішні стандарти, правові і нормативні акти, що стосуються діяльності організації, і практика їх застосування;
- співбесіда з найманими працівниками;
- результати бенчмаркінґу (еталоне тестування);
- патентна інформація;
- оцінки постачальників;
- оцінки замовників;
- інформація з Інтернету;
- маркетинґова інформація;
- документація замовників.

Внутрішньою інформацією і даними можуть бути:

- будь-які відомості, що отримують в результаті внутрішньої діяльності підприємства і є цінними для використання;
- результати експериментів;
- статистика та аналіз виявлених невідповідності;
- результати внутрішнього аудиту;
- звіти відділу маркетинґу;
- рейтинґи постачальників;
- процедури і методики;

- циркуляри і директиви.

Загальна корпоративна інформація використовується всередині корпорації і закрита для сторонніх організацій. Переважно така інформація розповсюджується закритою корпоративною системою Інтранет і може містити:

- корпоративні інструкції і директиви;
- звіти і листування з іншими організаціями;
- інформацію про корпоративні програми, стратегію, результати корпоративної діяльності;
- досвід організацій, рекомендований для розповсюдження;
- інформацію про замовників і партнерів;
- інформацію про публікації;
- інформацію про дискусії і семінари тощо.

Знання можуть бути класифіковані по наступних категоріях:

- поверхневі - знання про видимі взаємозв'язки між окремими подіями і фактами в наочної області;
- глибинні - абстракції, аналогії, схеми, що відображують структуру і процеси в наочної області.

База знань виконує наступні завдання:

- Виконує с*труктуризацію* інформації, даних і знань.
- *Формує і зберігає* правила для виробництва у які можуть бути зараховані спеціальні технології виконання окремих етапів процесу з рекомендованими типовими або експериментально перевіреними характеристиками.
- *Породжує* рішення, інформацію до роздумів, тобто знання співробітників, які повертаються в систему, доповнюючи і збагачуючи її.

Всю внутрішню інформацію, дані і знання можна розділити на **документовану** і **приховану** (tacit), яка міститься в голові працівника організації або в його особистих архівах.

Документована інформація (технологічна документація, документація контролінґу), правила (проектування, документообіг, процедури, інструкції), експериментальні дані, рекламації і дані про збої. Сьогодні знання набули чисто декларативної форми, тобто знаннями вважаються речення, записані на мовах подання знань, наближених до природніх і зрозумілих неспеціалістам. Існують десятки моделей (або мов) подання знань для різних наочних областей. Більшість з них може бути зведені до наступних класів: продукційні; семантичні мережі; фрейми; формальні логічні моделі і багато інших.

Існують десятки моделей (або мов) подання знань для різних наочних областей.

Більшість з них може бути зведені до наступних класів:

- продукційні;
- семантичні мережі;
- фрейми;
- формальні логічні моделі і багато інших.

19.2 Структура бази знань

Базу знань складають наступні елементи: база даних, база знань, блок розрахунків, блок виведення, логічних висновків та діагнозу, блок набуття знань, блок пояснень. Структурна схема бази знань наведено на рис.19.1 [15].

База даних містить планові, фактичні, розрахункові, звітні та інші постійні або оперативні показники. Наявність бази даних дозволяє використовувати стандартні процедури опрацювання файлів, що може різко скоротити витрати на їх підтримку.

База знань, відображає знання експерта про предметну сферу, способи аналізу фактів, що надходять, і методику висновків, тобто породження нових знань на підставі наявних знань та знань, що надійшли.

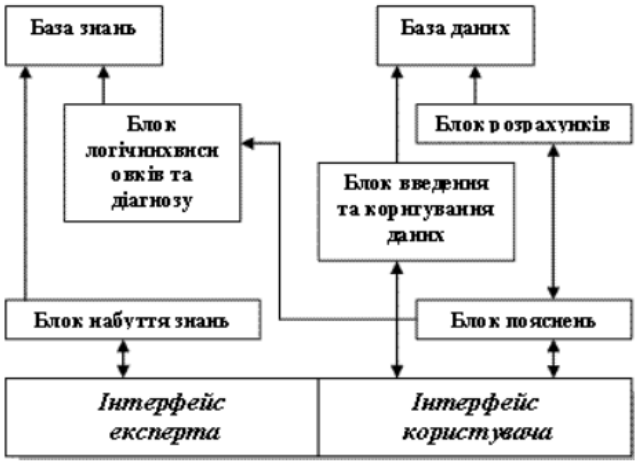


Рисунок 19.1 - Структурна схема бази знань

Блок розрахунків необхідний для супроводження прийняття управлінських рішень.

Блок введення та корегування даних має місце, якщо експертна система функціонує локально, тобто поза мережею передачі даних.

Блок логічних висновків і діагнозу є головним, оскільки за його допомогою користувач повинен встановити шлях виходу з економічної ситуації, що виникла. Для цього використовується факторний аналіз показників, результати якого потім використовуються для логічного аналізу і оформлення діагнозу. Діагноз можна одержати на підставі формалізації знань експертів за допомогою конструкції “якщо - то”. Якщо вдалось довести хоча б одне правило “якщо - то”, то це означає, що відомий перелік заходів, дій або процедур, які слід виконати для виходу із економічної ситуації, що створилась. Рецепти, як і діагнози, мають ієрархічний характер: кожному рівню діагнозу відповідає свій рецепт.

Блок набуття знань пов’язаний із проблемою навчання та самонавчання системи.

Блок пояснень організовує пояснення системи, чому вона дійшла до того чи іншого висновку. В експертних системах, заснованих на правилах, пояснення одержують простежуванням ще раз тих кроків міркування, що привели до даного висновку.

19.3 Стратегії здобуття знань

Існує безліч способів визначати поняття. Один з широко вживаних способів, заснований на ідеї **інтенсионала.**

Інтенсионал поняття - це визначення через поняття більш високого рівня абстракції з вказівкою специфічних властивостей. Цей спосіб визначає знання. Інший спосіб визначає поняття через перерахування понять нижчого рівня ієрархії або фактів, що відносяться до визначуваного. Це є визначення через дані, або екстенсионал поняття.

Існує декілька стратегій здобуття знань. Найбільш поширені:

- придбання;
- витягання;
- формування.

Під придбанням знань розуміється спосіб автоматизованої побудови бази знань за допомогою діалогу експерта і спеціальної програми (при цьому структура знань заздалегідь закладається в програму). Ця стратегія вимагає істотного попереднього опрацювання наочної області. Систем придбання знань дійсно набувають готові фрагменти знань відповідно до структур, закладених розробників систем. Більшість цих інструментальних засобів орієнтована на конкрет-

ні експертні системи з жорстко визначеною предметною областю і моделлю подання знань, тобто не є універсальними.

Термін витягання знань стосується безпосереднього живого контакту інженера по знаннях і джерела знань. Автори схильні використовувати цей термін як більш ємкий і такий, що більш точно виражає сенс процедури перенесення компетентності експерта через інженера по знаннях в базу знань експертної системи.

Термін формування знань традиційно закріпився за надзвичайно перспективною областю інженерії знань, яка займається розробкою моделей, методів і алгоритмів аналізу даних для здобуття знань і вчення, що активно розвивається. Ця область включає індуктивні моделі формування гіпотез на основі повчальних вибірок, вчення аналогічно і інші методи.

Виведення на знаннях. База знань – сукупність знань, що відносяться до деякої предметної області, формально представлених так, щоб на їх основі можна було здійснювати міркування. Бази знань найчастіше використовуються в контексті експертних систем, де з їх допомогою подаються навики і досвід експертів, зайнятих практичною діяльністю у відповідної області (наприклад, в медицині або в математиці). Зазвичай база знань є сукупністю правил виводу.

Не дивлячись на всі недоліки, найбільшого поширення набула продукційна модель подання знань. При використанні продукційної моделі база знань складається з набору правил. Програма, що управляє перебором правил, називається машиною виведення. Машина виведення (інтерпретатор правил) виконує дві функції: по-перше, перегляд існуючих фактів з робочої пам'яті (бази даних) і правил з бази знань і додавання (в міру можливості) в робочу пам'ять нових фактів, а по-друге, визначення порядку перегляду і вживання правил. Цей механізм управляє процесом консультації, зберігаючи для користувача інформацію про отримані висновки, і запрошує у нього інформацію, коли для спрацьовування чергового правила в робочій пам'яті виявляється недостатньо даних.

У переважній більшості систем, заснованих на знаннях, механізм виведення є невеликою за об'ємом програмою і включає два компоненти — один реалізує власне виведення, інший управляє цим процесом. Дія компонента виведення заснована на вживанні правила, яке називається modus ponens.

Правило modus ponens. Якщо відомо, що достеменне твердження *A*, *то* існує правило вигляду – «*Якщо A, то B*», тоді твердження *B* теж істинно. Правила спрацьовують, коли знаходяться факти, що задовольняють їх лівій частині: якщо достеменна посилання, то має бути достеменний і висновок. Компонент виведення повинен функціонувати навіть при недоліку інформації. Отримане рішення може і не бути точним, проте, система не повинна зупинятися через те, що відсутня будь-яка частина вхідної інформації. Компонент, що управляє, визначає порядок вживання правив і виконує чотири функції:

- **Зіставлення** — зразок правила зіставляється з наявними фактами.
- **Вибір** — якщо в конкретній ситуації може бути застосовано відразу декілька правив, то з них вибирається одне, найбільш відповідне по заданому критерію (вирішення конфлікту).
- **Спрацьовування** — якщо зразок правила при зіставленні збігся з якими-небудь фактами з робочої пам'яті, то правило спрацьовує.
- **Дія** — робоча пам'ять піддається зміні шляхом додавання в неї висновку що спрацював, правила. Якщо в правій частині правила міститься вказівка на будь-яку дію, то вона виконується (як, наприклад, в системах забезпечення безпеки інформації).

Інтерпретатор продукції працює циклічно. У кожному циклі він переглядає всі правила, щоб виявити ті, посилання яких збігаються з відомими на даний момент фактами з робочої пам'яті. Після вибору правило спрацьовує, його висновок заноситься в робочу пам'ять, і потім цикл повторюється спочатку. В одному циклі може спрацювати лише одне правило. Якщо декілька правил успішно зіставлені з фактами, то інтерпретатор виконує вибір по певному критерію єдиного правила, яке спрацьовує в даному циклі. Цикл роботи інтерпретатора схематично представлений на рис. 19.2.

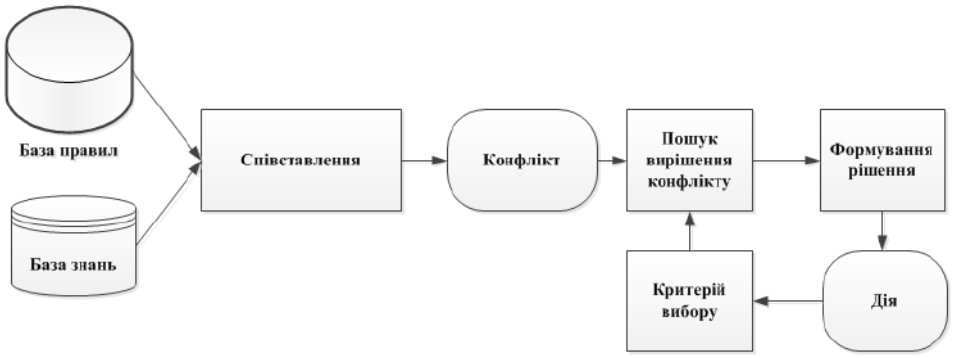


Рисунок 19.2 - Схема циклу роботи інтерпретатора

Інформація з робочої пам'яті послідовно зіставляється з посиланнями правил для виявлення успішного співставлення. Сукупність відібраних правил складає так звану конфліктну безліч. Для вирішення конфлікту інтерпретатор має критерій, за допомогою якого він вибирає єдине правило, після чого правило спрацьовує. Це виражається в занесенні фактів, які створюють висновок правила, в робочу пам'ять або в зміні критерію вибору конфліктуючих правил. Якщо ж по закінченню правила згадується назва якої-небудь дії, то воно виконується. Робота машини виведення залежить лише від стану робочої пам'яті і від складу бази знань. На практиці зазвичай враховується історія опрацювання, тобто поведінка механізму випаду в попередніх циклах. Інформація про поведінку механізму виведення запам'ятовується в пам'яті станів. Зазвичай пам'ять станів містить протокол системи.

Від вибраного методу пошуку, тобто стратегії виведення, залежить порядок використання і спрацьовування правил. Процедура вибору зводиться до визначення напряму пошуку і способу його здійснення. Процедури, що реалізовують пошук, зазвичай закладені в механізм виведення, тому в більшості систем інженери знань не мають до них доступу і, отже, не можуть в них нічого змінювати за власним бажанням.

При розробці стратегії управління виводом поважно визначити два питання:

- Яку точку в просторі станів прийняти як початкову? Від вибору цієї точки залежить і метод здійснення пошуку — в прямому або зворотному напрямі.
- Якими методами можна підвищити ефективність пошуку рішення?

Ці методи визначаються обраною стратегією перебору – глибину, ширину, по підзадачах або ін. При зворотному порядку виведення спочатку висувається деяка гіпотеза, а потім механізм виведення як би повертається назад, переходячи до фактів, намагаючись знайти ті, які підтверджують гіпотезу (рис. 19.3). Якщо вона виявилася правильною, то вибирається наступна гіпотеза, що деталізує першу і що є по відношенню до неї підціллю. Далі відшукуються факти, підтверджуючі істинність підпорядкованої гіпотези. Виведення такого типу називається керованим цілями, або керованим консеквентами. Зворотний пошук застосовується в тих випадках, коли цілі відомі і їх порівняно небагато.

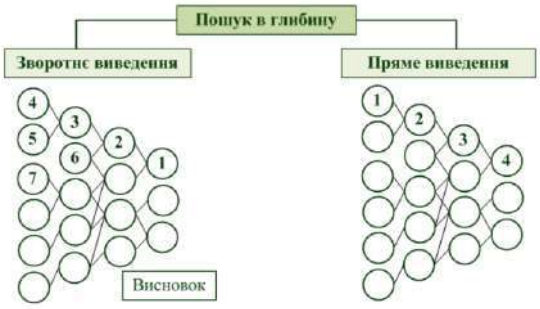


Рисунок 19.3 - Стратегії зворотному порядку виведення

У системах з прямим виведенням по відомих фактах відшукується висновок, який з цих фактів виходить (рис.19.4).

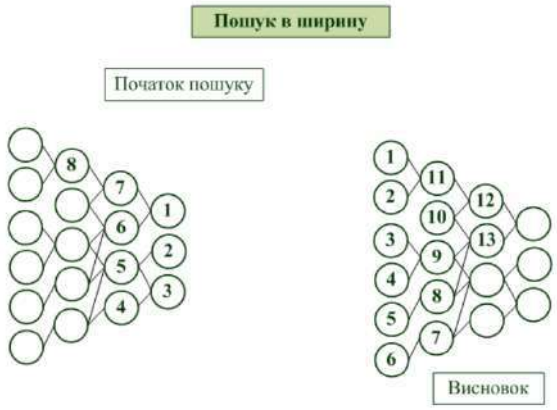


Рисунок 19.4 - Стратегії з прямим виведенням по відомих фактах

Якщо такі висновки вдається знайти, то вони заносяться в робочу пам'ять. Пряме виведення часто називають виведенням, керованими даними або керованими антецедентами. Існують системи, в яких вивід ґрунтується на поєднанні згаданих вище методів: зворотного і обмеженого прямого. Такий комбінований метод отримав назву циклічного.

Приклад. У системах, база знань яких налічує сотні правил, бажаним є використання стратегії управління виведенням, що дозволяє мінімізувати час пошуку рішення і тим самим підвищити ефективність виведення. До таких стратегій належать:

- пошук в глибину;
- пошук в ширину;
- розбиття на під задачі;
- альфа-бета.

При пошуку в глибину як чергова підмета вибирається та, яка відповідає наступному, детальнішому рівню опису завдання.

При пошуку в ширину, навпаки, система спочатку проаналізує всі симптоми, що знаходяться на одному рівні простору станів, навіть якщо вони відносяться до різних захворювань, і лише потім перейде до симптомів наступного рівня детальності.

Розбиття на підзадачі виконує виділення підзадач, вирішення яких розглядається як досягнення проміжних цілей на шляху до кінцевої мети. Прикладом, підтверджуючим ефективності розбиття на підзадачі, є пошук несправності в комп'ютері – спочатку виявляється підсистема (живлення, пам'ять і т. д.), що відмовила, що значно звужує простір пошуку. Якщо вдається правильно зрозуміти суть завдання і оптимально розбити її на систему ієрархічно зв'язаних цілей–підцілей, то можна домогтися мінімального шляху розв’язання в просторі пошуку.

Альфа-бета алгоритм дозволяє зменшити простір станів шляхом видалення гілок, неперспективних для успішного пошуку. Тому є видимими лише ті вершини, в які можна потрапити в результаті наступного кроку, після чого неперспективні напрями виключаються. Альфа-бета алгоритм знайшов широке вживання в основному в системах, орієнтованих на різні ігри, наприклад в шахових програмах.

19.4 Моделі знань

Моделювання – процес представлення досліджуваного об'єкту деякою послідовністю інших об'єктів або подань, що реалізовують ті або інші сторони об'єкту, що вивчається, з необхідною точністю. Модель завжди переслідує певну мету, і залежно від мети змінюється сама модель. Модель ніколи не відображає всю глибину об'єкту, що вивчається.

Розрізняють такі види моделей: модель предметної області, модель бази даних, модель бази знань.

Кожна модель зберігає знання про модельований фрагмент предметної області (інформаційній моделі) і виконує мовну функцію. Цей мовний компонент має велике значення при активізації моделі на ЕОМ. Перший етап побудови моделі пов'язаний з виявленням структури моделі на основі апріорної інформації, а другий етап пов'язаний з введенням в неї емпіричної інформації і є результатом узагальнення спостережень за реальним об'єктом.

При моделюванні знань і даних існує два аспекти: математичний і інструментальний. З точки зору математики модель може бути представлена безліччю відношень. Виникнення і розвиток інструментального аспекту обумовлене тим, що через відсутність в теорії банків інформації формальних методів побудови моделей ЕОМ прийняла на себе функції побудови моделей і стала інструментом побудови моделей. Ця модель призначена для опису на семантичному (смисловому) рівні предметної області і навколишнього середовища. Вона є засобом інтерпретації вмісту бази знань. З точки зору інструментального аспекту моделювання знань, інтерес представляють мовни засоби опису моделі.

Мови представлення знань можна розбити на три групи: логічні, реляційні, продукційні.

До логічних відносяться мови, засновані на численні висловів і предикатів. Основою реляційних мов є облік зв'язків (відношень) між предметами реального світу і їх властивостями.

Найбільш перспективною з мов даного типу є мова підстав на фреймах. Фрейм виступає як одиниця інформації, яка розглядає мінімальне необхідне число ознак в описі предмету, без яких цей предмет не існує.

Основою мови продукційного типу є поняття продукції, що складається з двох частин: умови і дії. Умовою є опис деякої ситуації, а дію визначає набір операцій, які необхідно здійснити, якщо вхідні параметри відповідають описаній ситуації.

Продукційна модель знань. Продукційна модель – модель, заснована на правилах, дозволяє представити знання у вигляді пропозицій типа – «Якщо (умова), то (дія)».

Під «умовою» (антецедент) розуміється деяке речення-зразок, по якому здійснюється пошук в базі знань, а під «дією» (консеквентом) - дії, що виконуються при успішному результаті пошуку (вони можуть бути проміжними, такими, що наступають далі як умови, і термінальними або цільовими, які завершують роботу системи). Найчастіше виведення на такій базі знань буває прямим (від даних до пошуку мети) або зворотним (від мети - до даних).

Дані – це вихідні факти, що зберігаються в базі фактів, на підставі яких запускається інтерпретатор правил, що перебирає правила з продукційної бази знань.

Продукційна модель найчастіше застосовується в промислових експертних системах. Вона приваблює розробників своєю наочністю, високою модульністю, легкістю внесення доповнень і змін і простотою механізму логічного виведення. Існує велика кількість програмних засобів, що реалізовують продукційний підхід (мова OPS 5; «оболонки» або «порожні» ЕС — EXSYS Professional, Kappa, ЕКСПЕРТ і ін.).

Семантична модель знань. Термін семантична означає «смислова», а сама семантика – це наука, що встановлює відношення між символами і об'єктами, які вони позначають, тобто наука, що визначає сенс знаків. Мережа – це орієнтований граф, вершини якого – поняття, а дуги – відношення між ними. Як поняття виступають конкретні об'єкти, а відношення – це зв'язки типа: «це» («АКО — A-kind-of» «is»), «має частиною» («has part»), «належить», «любить».

Характерною особливістю семантичних мереж є обов'язкова наявність трьох типів відношень:

клас – елемент класу;

властивість – значення;

приклад елементу класу.

Можна запропонувати декілька класифікацій семантичних мереж, пов'язаних з типами відношень між поняттями.

За кількістю типів відношень:

- Однорідні (з єдиним типом відношень).
- Неоднорідні (з різними типами відношень).

За типами відношень:

- Бінарні (у яких відношення зв'язують два об'єкти).
- N-арні (у яких є спеціальні відношення, що зв'язують більше двох понять).

Найчастіше в семантичних мережах використовуються такі відношення:

- зв'язки типу «частина – ціле» («клас – підклас», «елемент – безліч»);
- функціональні зв'язки, які визначуються зазвичай дієсловами – виконує, кількісні (більше, менше, дорівнює);
- просторові (далеко від, близько від, за, під, над);
- часові (раніше, пізніше, протягом);
- атрибутивні зв'язки (мати властивість, мати значення);
- логічні зв'язки (І, АБО, НЕ);
- лінгвістичні зв'язки і ін.

Проблема пошуку рішення в базі знань типу семантичної мережі зводиться до завдання пошуку фрагмента мережі, що відповідає деякій підмережі, що відображає поставлений запит.

На рис. 19.5 подана семантична мережа. Як вершини тут виступають поняття «ЕОМ – ПЕОМ - Ноутбук», «Toshiba Satellite C660-1EM», «Процесор» і «Материнська плата».

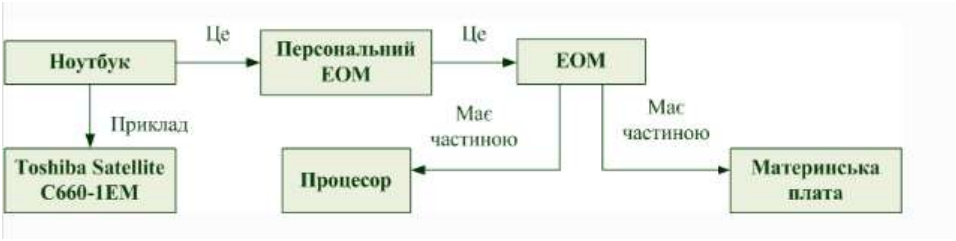


Рисунок 19.5 - Семантична мережа

Дана модель представлення знань була запропонована американським психологом Киллі-аном. Основною її перевагою є те, що вона більш за інших відповідає сучасним уявленням про організацію довготривалої пам'яті людини. Недоліком цієї моделі є складність організації процедури пошуку виведення на семантичній мережі. Для реалізації семантичних мереж існують спеціальні мережеві мови, наприклад NET [Цейтін, 1985], мова реалізації систем Simer+mir [Осипов, 1Э97] і ін. Широко відомі експертні системи, що використовують семантичні мережі як мову представлення знань, - PROSPECTOR, TORUS [Хейес-рот і ін. 19S7; Durkin, 1998].

Фрейм. Термін фрейм (від англійською «frame», що означає «каркас» або «рамка») був запропонований Марвіном Лі Мінським в 70-і роки як позначення структури знань для сприйняття просторових сцен. Ця модель, як і семантична мережа, має глибоке психологічне обґрунтування. Фрейм – це абстрактний образ для представлення деякого стереотипу.

Фреймом також називається і формалізована модель для відображення образу Розрізняють фрейми-зразки, або прототипи, що зберігаються в базі знань, і фрейми- екземпляри, які створюються для відображення реальних фактичних ситуацій на основі даних, що поступають.

Модель фрейму є досить універсальною, оскільки дозволяє відображувати все різноманіття знань про світ через такі елементи:

- фрейми-структури, що використовуються для позначення об'єктів і понять;
- фрейми - ролі;
- фрейми - сценарії;
- фрейми - ситуації .

Формально фрейм – це тип даних вигляду: F=<N, S1, S2, S3>,

- де N – ім’я об’єкту;
- S1 – множина слотів, що містять факти, визначаючі декларативну семантику фрейму;
- S2 – множина слотів, для зв’язків з іншими фреймами (каузальні, семантичні);
- S3 – множина слотів, для перетворення, що визначають процедурну семантику фрейму.

Структура фрейму може бути представлена як список властивостей:

(ІМ'Я ФРЕЙМА;
(ім'я 1-го слота; значення 1-го слота),
(ім'я 2-го слота: значення 2-го слота),
(ім'я N-го слота: значення N-го слота)).

При використанні табличної форми подання фрейму можна використовувати додаткові стовпці призначені для опису способу здобуття слотом його значення і можливого приєднання до того або іншого слота спеціальних процедур, що допускається в теорії фреймів.

Як значення слота може виступати ім'я іншого фрейму, так утворюються мережі фреймів. Існує декілька способів здобуття слотом значень у фреймі-екземплярі:

- за замовчуванням від фрейма-зразка (Default-значення);
- через спадковість властивостей від фрейму, вказаного в слоті АКО;
- по формулі, вказаній в слоті;
- через приєднану процедуру;
- явно з діалогу з користувачем;
- з бази даних.

Найважливішою властивістю теорії фреймів є запозичення з теорії семантичних мереж — так зване спадковість властивостей. Спадковість відбувається по АКО-зв'язках (A Kind of = це). Слот АКО вказує на фрейм більш високого рівня ієрархії, звідки неявно успадковуються, тобто переносяться, значення аналогічних слотів.

Фрейми утворюють ієрархію (рис. 19.6), остання породжує багаторівневу структуру, що описує або об’єкт за умови опису слотами лише властивостей об’єкту, або ситуацію чи процес за умови опису окремими слотами процедур, що під’єднані до фрейму та викликаються після його актуалізації.

Рисунок19.6 - Фреймова модель

Основні переваги фреймів як моделі подання знань є здатність відображати концептуальну основу організації пам'яті людини. Спеціальні мови представлення знань в мережах фреймів FRL (Frame Representation Language), KRL (Knowledge Representation Language), фреймова «оболонка» Карра і інші програмні засоби дозволяють ефективно будувати промислові ЕС. Широко відомі такі орієнтовані для фрейму експертні системи, як ANALYST, МОДІС, TRISTAN, Alteftld.

Логічна модель формальна. Основна ідея при побудові логічних моделей знань полягає в наступному – вся інформація, необхідна для вирішення прикладних завдань, розглядається як сукупність фактів і тверджень, що представляються як формули в деякій логіці. Знання відображуються сукупністю таких формул, а здобуття нових знань зводиться до реалізації процедур логічного виведення.

У основі логічних моделей знань лежить поняття формальної теорії, картежем, що задається у вигляді: S=<A, F, Ax, R>,

де A – множина базових символів (алфавіт);

F – множина, звана формулами;

Ax – виділена підмножина апріорі дійсних формул (аксіом);

R – кінцева множина відношень між формулами, так звані правила виведення.

Основні переваги логічних моделей знань:

- як «фундамент» тут використовується класичний апарат математичної логіки, методи якої досить добре вивчені і формально обґрунтовані;

- існують досить ефективні процедури виведення, в тому числі реалізовані в мові логічного програмування «Пролог»;
- у базах знань можна зберігати лише безліч аксіом, а всі останні знання отримувати з них за правилами виведення.

У логічних моделях знань слова, що описують суть предметної області, називаються термами (константи, змінні, функції), а слова, що описують відношення суті, – предикатами. Предикат – логічна N-арна пропозиціональна функція, визначена для предметної області і така, що набуває значень або істинності, або помилковості. Пропозиціональною називається функція, яка ставить у відповідність об'єктам з області визначення одне з значень істина/брехня. Предикат набуває значень «істина» або «брехня» залежно від значень вхідних в нього термів.

Спосіб опису предметної області, який використовується в логічних моделях знань, приводить до втрати деяких нюансів, властивих природному сприйняттю людини, і тому знижує описову можливість таких моделей. Складнощі виникають при описі «багатосортних» світів, коли об'єкти не є однорідними.

З метою подолання складнощів і розширення описових можливостей логічних моделей знань розробляються псевдофізичні логіки, логіки, що оперують з нечіткими знаннями, емпіричними кванторами, забезпечуючі індуктивні (від приватного до загального), дедуктивні (від загального до приватного) і традуктівні (на одному рівні спільності) виведення. Такі розширені моделі, що об'єднують можливості логічного і лінгвістичного підходів, прийнято називати логіко - лінгвістічними моделями предметної області.

19.5 Елементи експертних систем

Експертна система – це комплекс комп'ютерного програмного забезпечення, що допомагає людині приймати обґрунтовані рішення (рис.19.7). Експертні системи використовують інформацію, отриману заздалегідь від експертів – людей, які в якій-небудь області є кращими фахівцями. Всі експертні системи включають, принаймні, три основні елементи: базу знань, машину виводу і інтерфейс користувача.

База знань. База знань містить відомі факти, виражені у вигляді об'єктів, атрибутів і умов. Окрім описових уявлень про дійсність, вона включає вирази невизначеності – обмеження на достовірність факту. В цьому відношенні вона відрізняється від традиційної бази даних внаслідок свого символьного, а не числового або буквеного вмісту. При обробці інформації бази даних користуються заздалегідь визначеними логічними правилами Відповідно, база знань, що представляє вищий рівень абстракції, має справу з класами об'єктів, а не з самими об'єктами. База знань створюється людьми – консультантами.

Рисунок 19.7 – Елементи експертної системи

ПК виведення знань. Головним в експертній системі є механізм, що здійснює пошук в базі знань за правилами раціональної логіки для здобуття рішень. Ця машина виведення приводиться в дію при здобутті запиту користувача і виконує такі завдання:

- порівнює інформацію, що міститься в запиті користувача, з інформацією бази знань;
- шукає певну мету або причинні зв'язки;
- оцінює відносну визначеність фактів, ґрунтуючись на відповідних коефіцієнтах довіри, пов’язаних з фактом.

Як випливає з її назви, машина виведення призначена для побудови висновків. Її дія аналогічно міркуванням експерта-людини, яка оцінює проблему і пропонує гіпотетичні рішення. У пошуку цілей на основі запропонованих правил, машина виведення звертається до бази знань до тих пір, поки не знайде вірогідну дорогу до здобуття прийнятного результату.

Інтерфейс користувача. Завдання інтерфейсу користувача полягає в організації обміну інформацією між оператором і машиною виводу. Інтерфейс з використанням природної мови створює видимість довільної бесіди, застосовуючи повсякденні вирази в правильно побудованих пропозиціях. Очевидно, що ніж природніший такий інтерфейс, тим вище за вимогу до зовнішньої і оперативної пам'яті.

Отже, системи, що надають користувачеві максимум зручностей, витрачають більше ресурсів основної машини, доступних з видалених робочих станцій. Інструментальні засоби, призначені для роботи на персональних комп'ютерах, що мають обмежені можливості, неминуче приносять "дружність" в жертву ефективності. Інтерфейс прямого введення повинен уміти розпізнавати мову, або, принаймні, достатню кількість ключових слів і фраз, щоб уловлювати їх зв'язок з даною проблемою і її пропонованими рішеннями.

Аспект людський. У роботі з експертними системами беруть участь три групи людей. По-перше, адміністрація встановлює призначення експертної системи, обмежує предметну область, яку повинна охоплювати система, і точно визначає, які вигоди організація зможе отримувати з її використання. По-друге, фахівець із збору знань (інженер - когнітолог) збирає інформацію, необхідну для бази знань, порівнює відповідні дані і евристично організовує інформацію. По-третє, потенційний користувач вказує, як використовуватиметься система, якого роду проблеми належить вирішувати, і яким чином здійснюватиметься взаємодія програми з оператором. Також потрібний експерт (група експертів) у встановленої наочної області, для здобуття від нього знань, як у формі фактичної інформації, так і відносно аналітичних методів, які застосовуються для вирішення проблем в цій області.

Машинний аспект. Комп'ютерну частину системи представляють компоненти програмного забезпечення, які обробляють отриману інформацію про дійсність, закладену в символьному вигляді в базу знань. У базу знань поступають факти. Зв'язок між фактами представлений евристичними правилами – виразами декларативного знання про стосунки між об'єктами. Кожне таке правило має складову "якщо і компонент "те" (висновок-дія), які визначають прямий або зворотний причинно-наслідковий зв'язок. Дійсні твердження лише вірогідні, іншими словами, міра їх визначеності не завжди абсолютна. Кількісні коефіцієнти визначеності збільшують точність міркування експертних систем. Загальноприйнята схема полягає в тому, щоб варіювати рівень довіри від 0, що представляє мінімальну міру визначеності, до 100 – вища міра.

Контрольні питання.

1. Назвіть основні перетворення знань при машинній обробці?
2. Поясніть поняття інтенсионал та екстенсионал поняття.
3. Наведіть класифікацію знань.
4. Які способи здобуття знань ви знаєте?
5. Дайте визначення бази знань.
6. Які методи виведення ви знаєте?
7. Які стратегії управління виведенням ви знаєте?
8. Дайте визначення поняття «Експертна система».
9. Вкажіть склад експертної системи?
10. Назвіть основні функції експертної системи?
11. Поясніть поняття «Моделювання»?
12. Які є типи моделей баз знань?
13. Які є мови подання знань?
14. Визначення та основні характеристики продукційної мови.
15. Визначення та основні характеристики семантичної мови.
16. Визначення та основні характеристики мови фреймів.
17. Основні характеристики логічної мови подання знань.

Бібліографія

1. Хомопепко А. Д., Цыганков В. М., Мальцев М. Г. Базы данных: Учебник для высших учебных заведений / Под ред. А. Д. Хомопепко. — 6-е изд., доп. - СПб.: КОРОНА-Век, 2009. - 736 с.
2. Трофимов С. А. CASE-технологии: практическая работа в Rational Rose – М.: БИНОМ, 20014. – 348с.
3. Федотова Д.Э., Семенов Ю.Д., Чижик К.Н. CASE-технологии: Практикум. - М.: Горячая линия-Телеком, 2005.-160 с: ил.
4. Гольцман В. - MySQL 5.0. .- СПб.:Питер,2010.-253 с.:ил.
5. М. Кузнецов, И.Симдянов MySQL 5. СПб.: БХВ-Петербург,2010.-1007с.: ил
6. Джеффри Рихтер, Мартен ван де Боспурт. WinRT. Программирование на C# для профессионалов.- Диалектика, 2014.-386 с.
7. Рауль Чон, Иэн Хейкс, Рав Ахуджа. Начало работы с DB2 Express-C. Третье издание. IBM Corporation, 2010.-296с.
8. ШкрильА.А. Разработка клиент – серверних приложений в Delphi. СПб.: БХВ-Петербург, 2006.- 480с.: ил.
9. Ковягин А., Востриков, Мир Inter Base/FireBird/Yaffil-М.: КУДИЦ, 2002.-432с.
10. Грег Риккарди. Системы баз данных. Теория и практика использования в Internet и среде Java. Вильямс, 2001.
11. Литвин В.В. Методи та засоби інженерії даних та знань : навч. посіб. для студ. вищ. навч. закл. / В. В. Литвин // М-во освіти і науки, молоді та спорту України.– Л. : Магнолія 2006, 2012. – 248 с.
12. Пасічник В.В.Організація баз даних та знань: підручник для ВНЗ/ В.В. Пасічник, В.А. Резніченко.– К.: Видавнича група BHV,2006.-384с.