Лабораторна робота №1

Тема: Створення таблиці баз даних. Структура таблиці.

Мета: навчитися створювати найпростіші таблиці бази даних, організовувати індивідуальну структуру кожної таблиці, вносити зміни в структуру вже існуючих таблиць.

Хід роботи.

Для створення таблиці скористаємося утилітою ***Database Desktop***. Ця програма дозволяє створювати таблиці, змінювати структури, редагувати записи. Запустити ***Database Desktop*** можна наступними способами:

- натисканням кнопки ***Пуск / Borland Delphi 6 / Database Desktop***;
- в середовищі **Delphi** за допомогою команди ***Tools / Database Desktop*** головного меню.

Процес створення таблиці починається з команди ***File / New / Table***. При цьому необхідно вибрати формат (тип) таблиці та задати її структуру.(див. рис1.1)

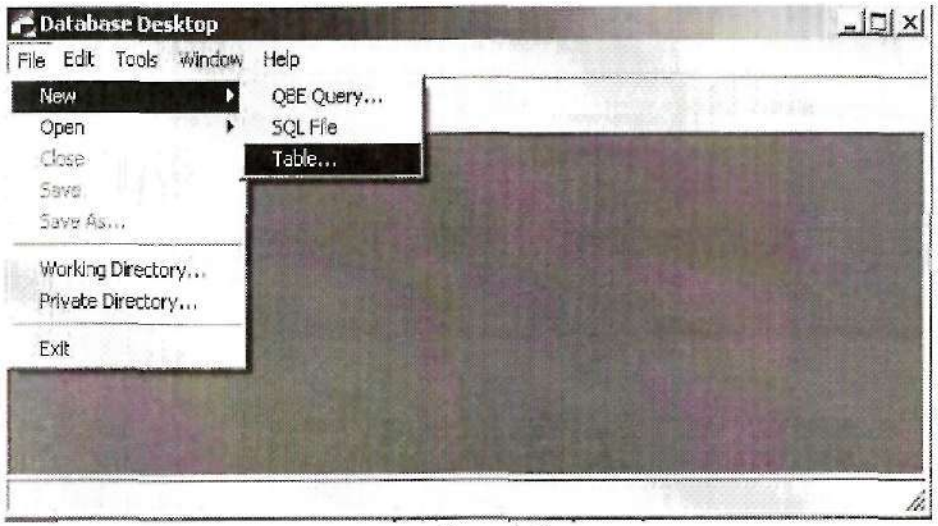
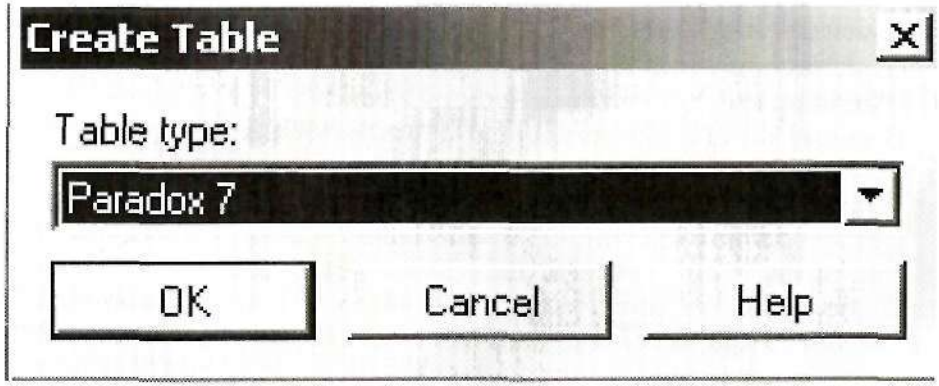


Рис. 1.1

В діалоговому вікні ***Create Table*** вибираємо тип таблиці.



За замовчуванням пропонується формат таблиці **Paradox** версії **7**. Після вибору типу таблиці з'являється вікно для визначення її структури. (див. рис 1.2)

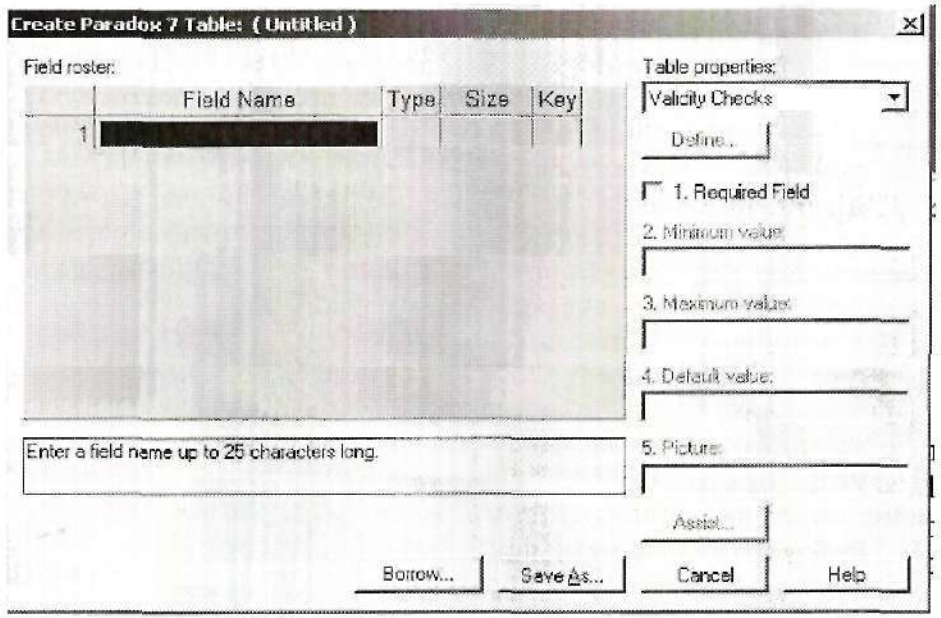


Рис. 1.2

В цьому вікні виконуються наступні дії:

- описання полів;
- визначення ключа;
- визначення індексів;
- визначення обмежень на значення полів;
- визначення обмежень посилальної цілісності;

- визначення паролів;
- визначення мовного драйвера;
- визначення таблиці для вибору значень.

В цьому списку обов'язковою є лише перша дія, тобто кожна таблиця має мати хоча б одне поле. Інші виконуються лише при необхідності.

Центральною частиною вікна визначення структури є список полів ***Field rosters***. Ім'я поля задається в стовпчику ***Field Name***. Воно вводиться за правилами, які встановлені для обраного формату таблиць (для запобігання виникнення помилок бажано використовувати латинські літери). Стовпчик ***Type*** визначає тип поля. Його можна визначити, безпосередньо вказавши відповідний символ, або вибрати зі списку, який розкривається при натисканні клавіші <***Space***> в стовпчику ***Type***. В стовпчику ***Size*** задається розмір. Але необхідність його визначення залежить від типу поля. Якщо поле є ключовим, то в значення поля ***Key*** повинно бути "*". Для цього потрібно перевести курсор у відповідну позицію і натиснути довільну алфавітно-цифрову клавішу. При повторному натисканні відмітка "*" знімається.

Для того, щоб додати нове поле, необхідно перевести курсор вниз на новий рядок. Вставка нового рядка виконується за допомогою клавіші <***Insert***>, знищення – комбінацією клавіш <***Ctrl***>+<***Del***>.

Для виконання інших дій по визначенню структури використовуємо комбінований список ***Table properties***, який містить наступні пункти:

- ***Secondary Indexes*** – індекси;
- ***Validity Checks*** – перевірка правильності вводу значень (вибирається за замовчуванням);
- ***Referential Integrity*** – посилальна цілісність;
- ***Password Security*** – паролі;
- ***Table Language*** – мова таблиці (мовний драйвер);
- ***Table Lookup*** – таблиця вибору;
- ***Dependent Tables*** – підлеглі таблиці.

Після вибору будь-якого пункту цього списку в правій частині вікна з'являються відповідні елементи, за допомогою яких виконуються подальші дії.

Визначення індексів. Задати індекс означає визначити:

1. Склад полів.

2. Параметри.
3. Ім'я.

Ці елементи встановлюються або змінюються під час виконання операції створення, зміни та знищення індексу.

Для виконання операцій, пов'язаних із визначенням індексів, необхідно обрати пункт ***Secondary Indexes*** комбінованого списку, при цьому під списком з'являються кнопки ***Define*** (Визначити) та ***Modify*** (Змінити), список індексів та кнопка ***Erase*** (Знищити). В списку індексів виводяться імена створених індексів. Для таблиць формату **Paradox** індекс також називається вторинним індексом.

Створення нового індексу починається з натискання кнопки ***Define***, яка завжди є доступною. Це веде до появи вікна ***Define Secondary Indexes***, в якому задаються склад полів та параметри індекса.(див. рис. 1.3)

В списку ***Fields*** (Поля) вікна виводяться імена всіх полів таблиці, включаючи й ті, які не можна включити в склад індексу (напр. графічне, або поле коментарів). У списку ***Indexed Fields*** (Індексні поля) містяться поля, які включаються в склад індексу, що створюється. Поле не може бути повторно включене в склад індексу, якщо воно вже обрано.

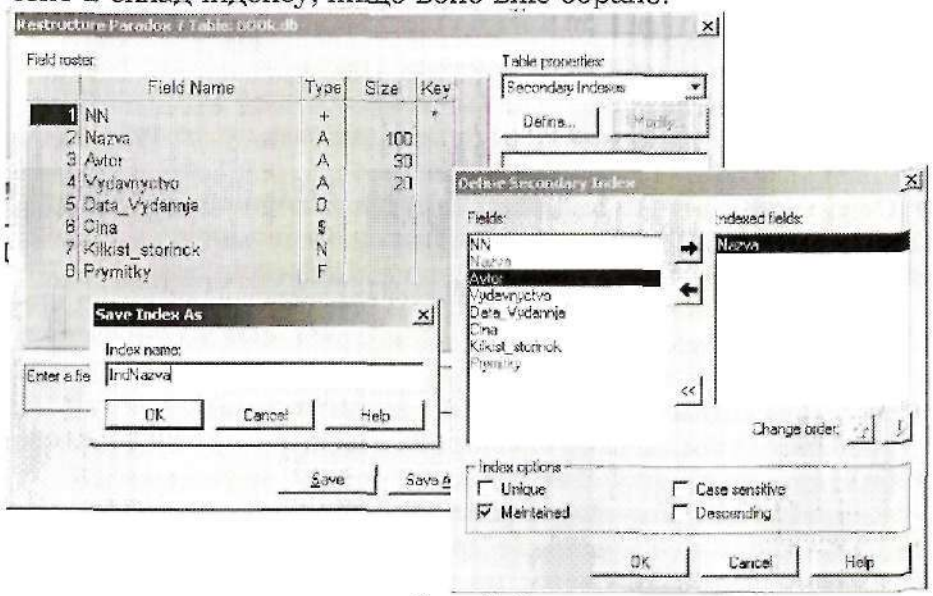


Рис. 1.3

Зауваження. При роботі із записами індексні поля обробляються в порядку слідування цих полів в складі індексу. Це треба враховувати, коли вказується порядок полів в індексі. Змінити порядок слідування полів в індексі можна за допомогою кнопок із зображенням вертикальних стрілок, що мають загальну назву ***Change order***. Для зсуву поля (полів) необхідно його (їх) виділити і натиснути потрібну кнопку.

Перемикачі, які розміщені в нижній частині вікна визначення індексу, дозволяють вказати наступні параметри індексу:

Unique - індекс потребує для полів, що його складають, унікальних значень.

Maintained – якщо таблиця відкрита, індекс автоматично не модифікується.

Case sensitive – для полів рядкового типу враховуються регістри символів.

Descending – сортування виконується в порядку зменшення значень.

Так як для таблиць dBase немає ключів, то для них використання параметра *Unique* є єдиною можливістю забезпечити унікальність записів на рівні організації таблиці, не застосовуючи програмування.

Після задання складу індексних полів з'являється вікно ***Save Index As***, в якому вказується ім'я індексу. Після повторного натискання сформований індекс додається до таблиці, і його ім'я з'являється в списку індексів.

Створений індекс можна змінити, визначивши новий склад полів, параметрів та імені індексу. Зміна індексу не відрізняється від процесу його створення. Після виділення індексу і натискання кнопки ***Modify*** знову відкривається вікно визначення індексу. При натисканні кнопки **<OK>** з'являється вікно збереження індексу, що містить ім'я індексу, який змінюють, і яке можна виправити, або лишити таким, як було.

Для знищення індексу його треба виділити в списку індексів і натиснути кнопку ***Erase***.

Кнопки ***Modify*** та ***Erase*** доступні, якщо тільки індекс обрано в списку.

Визначення обмежень на значення полів. Для виконання необхідних операцій вибираємо пункт ***Validity Checks***. (див. рис1.4)

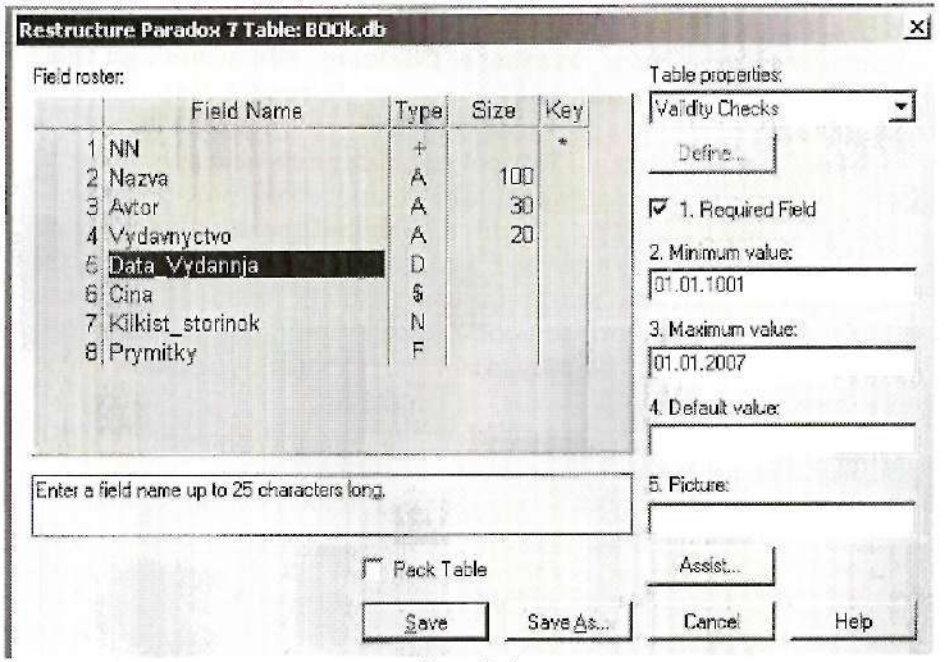


Рис 1.4

В цьому вікні можна задати обов'язковість вводу значень, встановити мінімальне, максимальне та значення за замовчуванням. В полі ***Picture*** можна задати маску(шаблон) для деякого поля. Це можна задати вручну в даному полі, або при натисненні кнопки ***Assist*** в вікні ***Picture Assistance***. Даний помічник створення масок містить стандартні шаблони, дозволяє їх редагувати та створювати нові.

Визначення посилальної цілісності. Поняття ***посилальної цілісності*** належить до зв'язаних таблиць і проявляється в наступних варіантах взаємодії таблиць:

1.Забороняється зміна поля зв'язку або знищення запису головної таблиці, якщо для неї є записи в підлеглій таблиці.

2.При знищенні запису в головній таблиці, автоматично знищуються відповідні до неї записи в підлеглій таблиці.

Для завдання обмежень посилальної цілісності необхідно обрати пункт ***Referential Integrity*** комбінованого списку ***Table properties***. При цьому з'являються кнопки ***Modify, Define*** та ***Erase.*** Для створення посилальної цілісності треба натиснути кнопку ***Define.*** З'являється вікно ***Referential Integrity.*** В списку ***Fields*** треба обрати поле зв'язку і натиснути кнопку із стрілкою вправо і перевести його в список

Child fields. В списку ***Table*** вказана головна таблиця і імена обираються із програми **Database Desktop.** Після вибору таблиці і натискання кнопки <←–> в список ***Parent's key*** автоматично заносяться ключові поля основної таблиці. Параметри посилальної цілісності обираються перемикачами. Група ***Update rules*** (правила зміни) визначають вигляд взаємодії таблиць під час зміни в головній таблиці. Перемикач ***Cascade*** встановлює режим каскадного знищення записів в підлеглій таблиці при видаленні відповідного запису головної таблиці. Перемикач ***Prohibit*** встановлює режим заборони зміни поля зв'язку або видалення запису головної таблиці, для якої є записи в підлеглій таблиці.

Прапорець ***Strict referential integrity*** встановлює захист таблиць від модифікацій із використанням старих версій програми, які не підтримують посилальну цілісність. Після встановлення потрібних режимів та натискування кнопки **<OK>** з'являється вікно ***Strict referential integrity As***, в якому треба вказати ім'я умови. Для таблиць формату **Paradox** умови посилальної цілісності іменуються.

Визначення паролів. Паролі дозволяють визначати права доступу користувачів до таблиці БД.

Якщо для таблиці задати пароль, то він буде автоматично запитуватися під час кожної спроби відкрити таблицю. Для виконання операцій, пов'язаних із визначенням пароля, потрібно обрати рядок ***Password Security*** в комбінованому списку ***Table properties***. З'являються кнопки ***Modify, Define.*** Натиснувши на кнопку ***Define,*** з'являється вікно ***Password Security***, в якому задається пароль. Пароль вводять два рази в полях ***Master password*** (головний) і ***Verify master password*** (перевірка головного пароля). При натисканні кнопки **<OK>** обидва значення звіряються. Якщо вони співпадають, то пароль приймається. Коли пароль визначено, кнопка ***Define*** блокується, стає доступною кнопка ***Modify*** зміни пароля і з'являються кнопки ***Change*** і ***Delete***, а поля введення блокуються. Натискуванням кнопки ***Delete*** пароль знищується і його можна ввести знову. Натискуванням ***Change*** редактори розблоковуються і значення паролю можна змінити в обох редакторах. А ***Change*** змінюється на ***Revert*** (повернення) і її повторне натискання повертає значення паролю до початку

редагування. Кнопка ***Auxiliary Password*** – завдання додаткових паролів. (див. рис. 1.5)

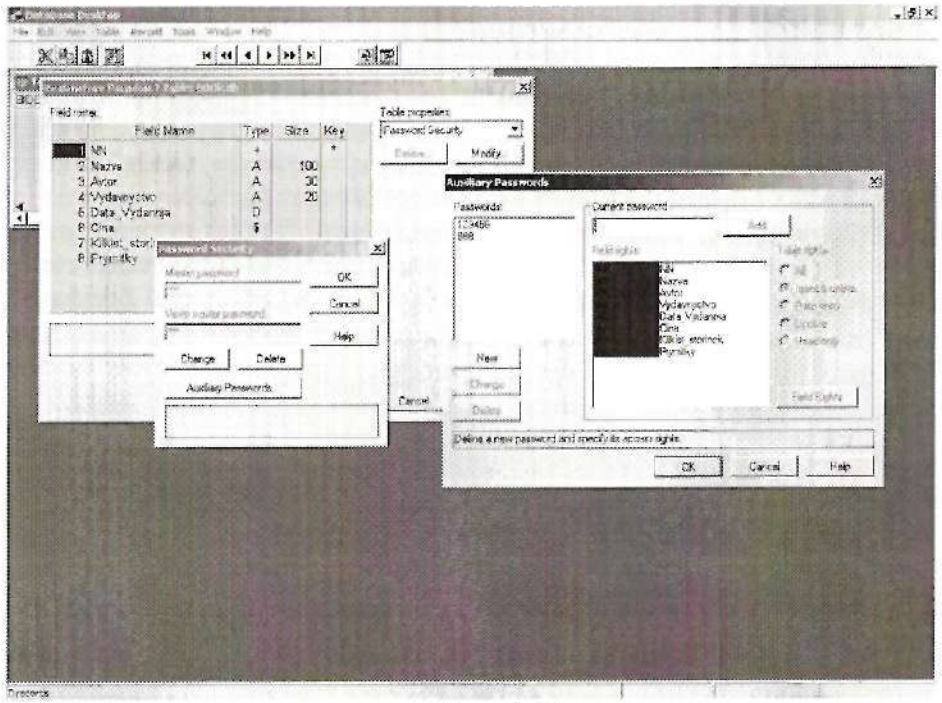


Рис. 1.5

Визначення мовного драйверу. Вибір мовного драйверу здійснюється за допомогою пункту ***Table Language***. Для коректної роботи таблиці із кирилицею, у потрібно змінити поточний параметр на ***Pdox ANSI Cyrillic***.

Визначення таблиці для вибору значень. У випадках, коли деяке поле може приймати значення із деякого набору, корисно помістити даний набір у деяку таблицю і значення поля вибирати із записів іншої таблиці. Поля таблиці вибору використовують для формування набору допустимих значень. Якщо для поля задано таблицю вибору, то в нього можна ввести тільки значення, що містяться в таблиці Для використання операторів, пов'язаних з полями вибору, призначений пункт ***Table Lookup***.(див. рис1.6)

При цьому під списком стає доступною кнопка ***Define,*** натискання на яку відкриває вікно ***Table Lookup***.

В списку ***Fields*** виводяться імена всіх полів таблиці. Ім'я поля, для якого задається таблиця вибору, відображається в області ***Field name***. Для його визначення треба виділити в списку ***Fields*** потрібне поле і натиснути →. В списку ***Lookup table*** задається таблиця вибору, ім'я якої з'являється в редакторі списку. В списку ***Drive (or Alias)*** задається диск або псевдонім, які визначають розміщення таблиці вибору. Назва каталогу, таблиці якого містяться в списку ***Lookup table***, відображаються справа від назви списку ***Drive (or Alias)***. Після завдання таблиці вибору натисканням кнопки ← переводить ім'я її першого поля, значення якого будуть використані для формування набору допустимих даних, в область ***Lookup field*** (поле вибору). Список, сформований на основі значень поля (полів) таблиці вибору, з'являється при натисканні комбінації клавіш <***Ctrl***>+<***Space***> в полі, яке редагується.

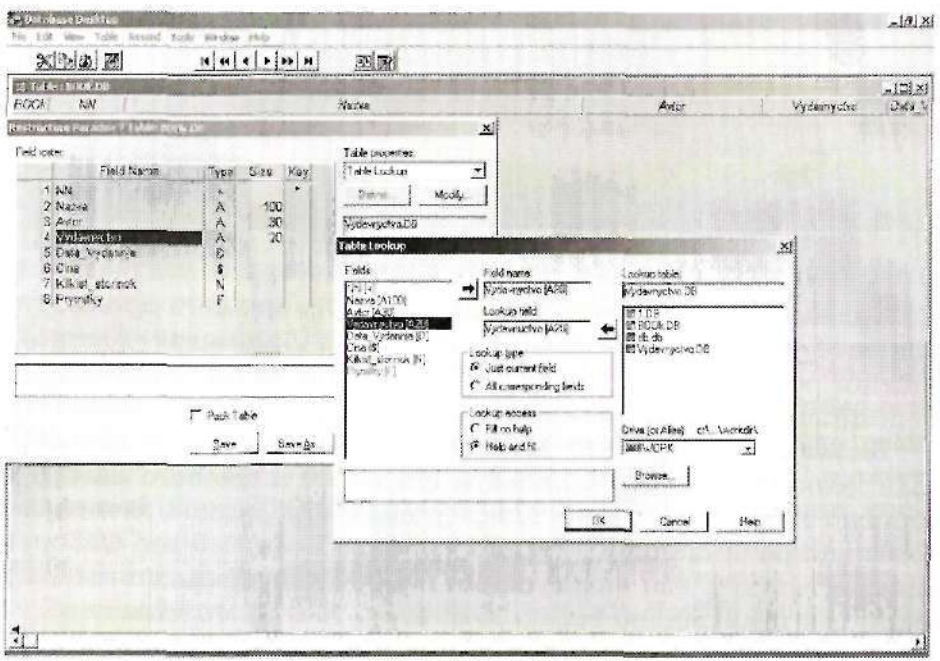


Рис. 1.6

За допомогою ***Lookup type*** (Тип вибору) – можна задати спосіб взаємодії обох таблиць. ***Just current field*** – таблиця вибору задасться тільки для поля, вказаного в області ***Field name***.

Lookup access (Доступ до таблиці вибору) – визначає, яким чином користувач користується значеннями з таблиці вибору. ***Fill no help*** – вставка без допомоги.

Для головної таблиці можна переглянути список підлеглих таблиць вибором ***Dependent Table*** комбінованого списку.(див. рис.1.7)

Змінити структуру таблиці можна за допомогою пункту меню ***Table \ Restructure***. При зміні структури з таблицею не повинні працювати інші програми(в тому числі і ***Delphi***). Зміна імені таблиці повинна виконуватись із середовища ***Database Desktop*** (***Save as***..), а не із середовища **Windows**, тому що інформація про назву таблиці використовується всередині її файлів.

Рис. 1.7

Контрольні завдання:

Після засвоєння матеріалу лабораторної роботи слід виконати наступні завдання:

1. Створити таблицю бази даних з кількістю полів більше десяти, при цьому використати не менше шести різних типів полів (роботу з цією ж таблицею буде продовжено в подальшому).
2. Задати потрібні індекси по одному, а також по декількох полях.
3. Задати потрібні обмеження на поля.
4. Задати паролі для доступу до таблиці різних категорій користувачів.
5. Вибрати коректний мовний драйвер.
6. Для деякого поля створити таблицю для вибору значень.
7. Занести в таблицю 21 запис.
8. Оформити короткий звіт про виконану роботу, внести в нього основні теоретичні відомості.

Лабораторна робота №2

Тема: Створення програмного додатку для обробки даних таблиці БД.

Мета роботи: ознайомитися з візуальними та невізуальними компонентами **Delphi**, що призначені для обробки даних таблиці, навчитися використовувати їх для створення найпростішого додатку.

Хід роботи.

Якщо таблиці БД розміщені не в директоріях по замовчуванню, потрібно створити **псевдонім** (alias) – шлях до таблиці. Псевдонім вказує місцезнаходження файлів БД і являє собою спеціальне ім'я для позначення каталогу. Використання псевдонімів значно полегшує перенесення файлів БД в інші каталоги і на інші комп'ютери. Для його створення перед викликом пункту меню ***Object \ New...*** Адміністратора ***BDE*** (***Пуск / Borland Delphi 6 / BDE Administrator***) потрібно вибрати закладку ***Database*** в лівій частині вікна, в противному випадку команда ***New*** буде недоступна. Після завдання цієї команди з'явиться діалогове вікно ***New Database Alias*** (Новий псевдонім БД), в якому треба обрати тип драйвера. Для локальних таблиць Paradox вибираємо тип **STANDARD**. Новий псевдонім автоматично отримає назву **STANDARD1** і параметри за замовчуванням. Псевдонім для роботи з локальними БД має три параметри:

DEFAULT DRIVER – вказує формат таблиць БД (за замовченням має формат PARADOX).

ENABLE BCD – вказує на необхідність переводу чисел в формат BCD, що дозволяє більш точно виконувати обчислення, але зменшує швидкість їх виконання. За замовченням має значення False і, відповідно, формат BCD не використовується.

PATH – вказує розташування (каталог) БД. Після створення псевдоніму шлях не визначено і користувач має встановити його самостійно.

Шлях задаємо параметром "***Path***" вручну, або використовуючи кнопку "**...**". (див. рис 2.1)

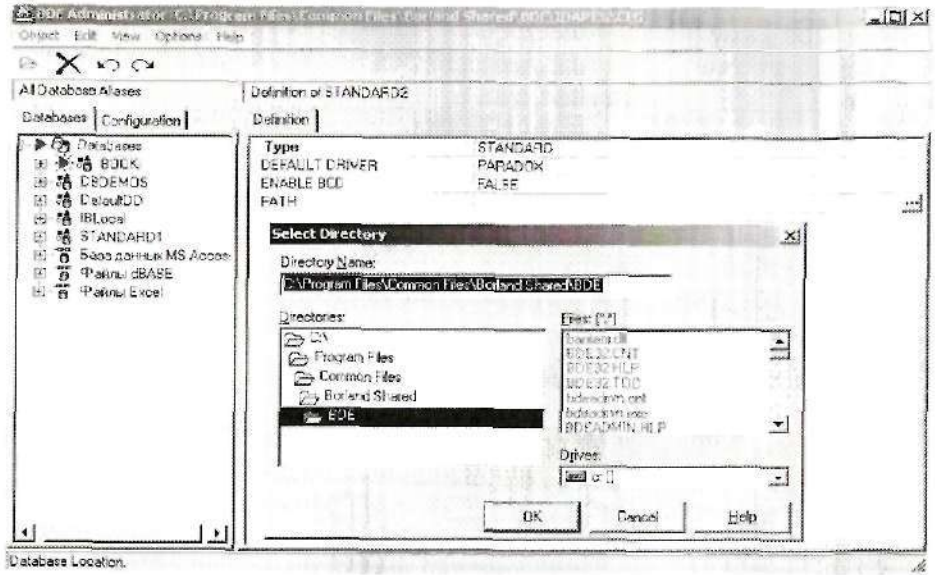


Рис 2.1

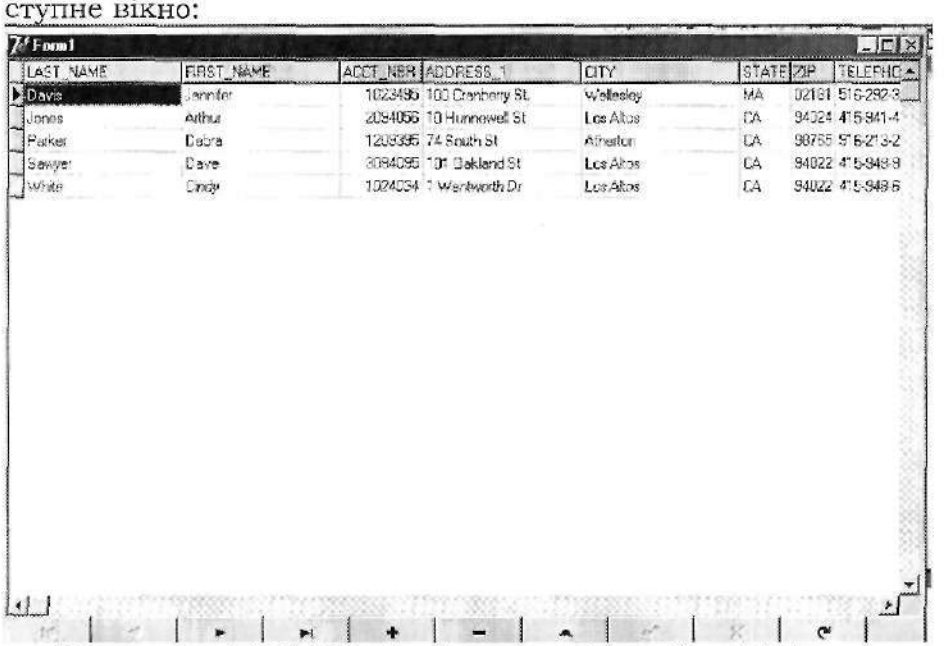
При створенні найпростішого проекту, що використовує механізм доступу **BDE** і дозволяє пересуватися по записах БД, переглядати та редагувати поля, знищувати записи з таблиці та додавати нові, головною задачею є конструювання форми. На формі розміщені компоненти ***Table1*** (закладка **BDE**), ***DataSourse1*** (закладка ***DataAccess***), ***DBGrid1*** і ***DBNavigator1*** (закладка ***Data Controls***).

Компонент ***Table1*** забезпечує взаємодію з реальною базою даних. Для зв'язку з потрібною таблицею треба встановити відповідне значення властивостям ***DatabaseName***, яка вказує шлях до БД, і ***TableName***, що визначає ім'я таблиці. Після завдання таблиці для відкриття набору даних властивості ***Active*** має бути встановлене значення ***True***. Компонент ***DataSourse1*** є проміжковою ланкою між компонентом ***Table1*** і візуальними компонентами ***DBGrid1*** і ***DBNavigator1***, за допомогою яких користувач взаємодіє з цією таблицею. На компонент ***Table1***, з яким пов'язаний компонент ***DataSourse1***, вказує властивість ***DataSet*** останнього. В таблиці 1 наведено компоненти, які використовуються для роботи з БД, їх основні властивості та значення цих властивостей.

Таблиця 1. Значення властивостей компонент.

Компонент	Властивості	Значення
Table1	*DataBaseName* *TableName* *Active*	*dbdemos* *Clients.dbf* *True*
DataSource1	*DataSet*	*Table1*
DBGrid1	*DataSource*	*DataSource1*
DBNavigator1	*DataSource*	*DataSource1*

Після встановлення необхідних параметрів, отримаємо наступне вікно:



Компонент ***DBGrid1*** відображає вміст таблиці БД у вигляді сітки, в якій стовпчики відповідають полям, а рядки – записам. За замовчуванням користувач може переглядати та редагувати дані. Компонент ***DBNavigator1*** дозволяє здійснювати пересування по таблиці, редагувати, вставляти та знищувати записи. Навігатор містить кнопки, які забезпечують виконання різних операцій з набором даних шляхом автоматичного виклику відповідних методів. *Склад кнопок* визначає властивість ***VisibleButtons*** типу ***TButtonSet***, що приймають комбінації наступних кнопок:

nbFirst - перейти до першого запису;

nbPrior - перейти до попереднього запису;
nbNext - перейти до наступного запису;
nbLast - перейти до останнього запису;
nbInsert - вставити новий запис;
nbDelete - видалити поточний запис;
nbEdit - редагувати поточний запис;
nbPost - затвердити результат зміни запису;
nbCancel - відмінити зміни в активному запису;
nbRefresh - відновити інформацію в наборі даних.

Для переміщення по таблиці використовуємо "↑", "↓". Для вставки записів використаємо клавішу <***Insert***>, для знищення – комбінацію клавіш <***Ctrl***>+<***Del***>.

В процесі вставки записів дані заносяться безпосередньо у таблицю. Даний факт не є зручним. Для спрощення процесу вставки використаємо ряд компонентів з вкладки ***Data Controls***. Це є компоненти ***DBText, DBEdit, DBMemo, DBImage, DBListBox, DBComboBox, DBCheckBox, DBRadioGroup, DBLookupListBox, DBLookupComboBox, DBRichEdit, DBCtrlGrid, DBChart***. Ці компоненти використовуються для відображення і роботи з полями таблиці відповідних типів. Виберемо деякі з цих компонент та розташуємо їх на формі.

Логічне поле може містити одне з двох значень: ***True*** і ***False***. Для відображення зміни логічного поля можна використовувати редактор ***DBEdit***. Але зручніше виконувати ці дії за допомогою незалежного перемикача ***DBCheckBox***, який дозволяє «включити» та «виключити» значення логічного поля. Компонент ***DBCheckBox*** виглядає на екрані як квадратик з текстовим заголовком. Якщо в ньому знаходиться галочка, то пов'язане з цим квадратиком логічне поле активного запису містить значення ***True***. Якщо квадратик пустий, то логічне поле поточного запису містить значення ***False***.

Іноді виникає необхідність введення або відображення в полі одного із значень, яке входить в склад фіксованого набору. З цією метою зручно використовувати компонент ***DBRadioGroup***, які являють собою групу перемикачів, із яких в кожний момент часу може бути включений (обраний) тільки один. Залежні перемикачі також називають перемикачами; в формі вони відображаються у вигляді кружка з текстовим надписом.

Керування кількістю та назвою перемикачів виконується за допомогою властивості ***Items*** типу ***TStrings***, яка до-

зволяє отримати доступ до окремих перемикачів в групі. Ця властивість містить рядки, які відображаються як заголовки перемикачів.

Властивість ***Values*** типу ***TStrings*** містить список значень поля, на які мають реагувати перемикачі групи. Керування цим списком виконується аналогічно керуванню списком ***Items***.

Для відображення поля активного запису і вибору для цього поля нового значення, служать списки – компоненти ***DBListBox, DBComboBox, DBLookupComboBox, DBLookupListBox***.

Компоненти ***DBListBox*** та ***DBComboBox*** дозволяють користувачеві обирати один із рядкових елементів. Під час вибору потрібного рядку простого списку ***DBListBox*** значення, що в ньому міститься, автоматично заноситься в поле, з яким пов"язаний цей список. В доповнення до цього комбінований список ***DBComboBox*** дозволяє вводити довільне значення.

Компоненти ***DBLookupComboBox*** та ***DBLookupListBox***, які також призначені для вибору значень в списку або безпосереднього введення значення в поле редагування (тільки для ***DBLookupComboBox***). Компонент ***DBLookupComboBox*** (або ***DBLookupListBox***) зв'язується з полем «свого» набору даних через властивості **DataSource** і ***DataField***, а список формується за допомогою властивостей ListSource (типу ***TDataSource***) і ***DataField*** (типу ***String***), які вказують на другий набір даних і його поле, яке використовується для заповнення списку.

У властивості ***KeyField*** типу ***String*** вказується поле другого набору даних, значення якого заноситься в поле, яке пов'язане з компонентом списку.

Зв'яжемо кожну компоненту з відповідним полем вже створеної (заповненої) таблиці БД. Для цього використовуємо властивість ***DataSourse*** відповідних компонентів. Її значення повинно бути рівним ***DataSourse1***. За допомогою властивості ***DataField*** встановлюємо зв'язки даних компонентів із відповідними їм полями (із списку, який розкривається, вибираємо назву поля, з яким зв'язуємо компонент). Тобто маємо ситуацію зображену на рисунку 2.2.

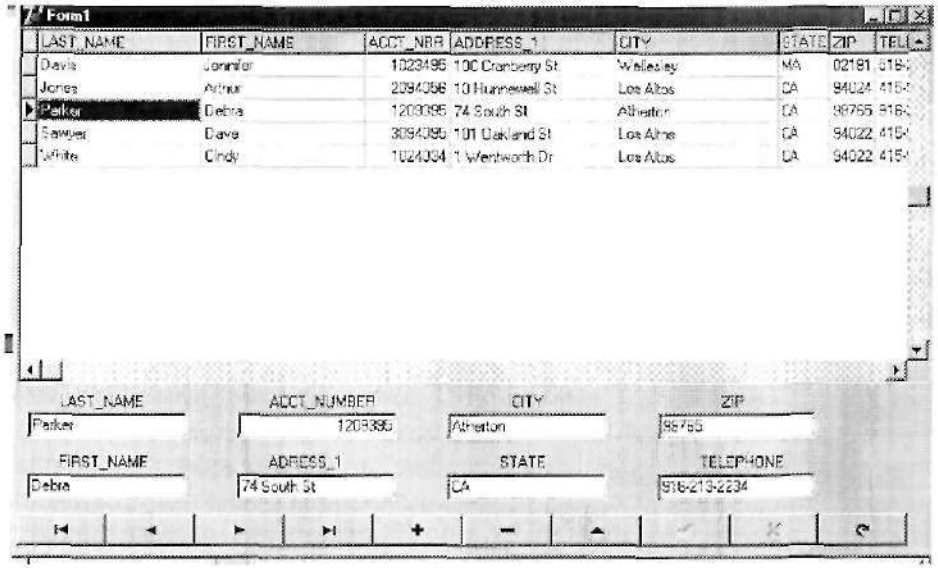


Рис. 2.2

Надамо значення полям, використовуючи вибрані компоненти. Для цього натиснемо кнопку «+» на навігаторі або клавішу <***Insert***>. Потім заповнюємо відповідні поля після чого натиснемо кнопку ***nbPost*** (кнопка з символом «галочка»).

Створення модуля даних. Модуль, як і форма, є своєрідним контейнером для своїх невізуальних компонентів. Він дозволяє:

1. Відокремити керування БД від обробки даних.
2. Створити модуль, який сумісно можна використовувати декількома додатками.

Для нього створюється модуль коду з розширенням *.pas*. Додавання модуля даних до проекту виконується шляхом вибору об'єкту ***Data Module*** на сторінці ***New*** сховища об'єктів, доступ до якого здійснюється командою ***File/New/Others*** головного меню **Delphi**. Для того, що би зв'язати модуль з проектом, треба клацнути на кнопці ***Add file to progect*** панелі інструментів.

В процесі звернення до компонентів, які містяться в модулі даних, для них вказують складове ім'я, в яке крім компоненти, входить також ім'я модуля даних:

```
procedure TForm1.FormCreate(Sender:TObject);
begin
```

Data Module2.Table1.TableName:="clients.dbf";
Data Module2.Table1.DatabaseName:="dbdemos";
Data Module2.DataSource1.DataSet:= Data Module2.Table1;
DBGrid1. DataSourse:= Data Module2.DataSource1;
DBNavigator1. DataSourse:= Data Module2.DataSource1;
Data Module2.Table1. Active:= True;
end;

Для забезпечення можливості доступу до компонентів модуля даних в модулі форми, в список ***uses*** розділу ***implementation*** необхідно включити посилання на модуль даних

uses unit2;

Контрольні завдання:

Після засвоєння матеріалу лабораторної роботи слід вміти:

1. Задати шлях (аліас) до директорії, в якій розташовані таблиці.
2. Створити найпростіший програмний додаток, використовуючи компоненти Delphi.
3. Встановити зв'язки між обраними компонентами.
4. Використовувати для роботи з таблицею компоненти ***DBText, DBEdit, DBMemo, DBListBox, DBComboBox, DBCheckBox.***
5. Модифікувати таблицю з даними за допомогою розглянутих компонент.
6. Створити модуль даних.
7. Оформити короткий звіт про виконану роботу, внести в нього основні теоретичні відомості.

Лабораторна робота №3

Тема: Інструментальні засоби та компоненти для роботи з базами даних. Набори даних.

Мета роботи: вивчити поняття набору даних, ознайомитися з основними властивостями компонентів ***Table, DBGrid.***

Хід роботи.

Для роботи з наборами даних (НД) будемо використовувати компонент ***Table***. Використаємо проект з попередньої лабораторної роботи.

Кількість записів в таблиці визначає властивість ***RecordCount*** типу ***Longint***. Циклом від 1 то ***RecordCount*** виконується перебір всіх записів таблиці. Метод ***First*** виконує перехід на перший запис таблиці, для переходу до наступного запису використаємо метод ***Next***. Для таблиці Paradox використовуючи властивість ***RecNo*** типу ***Longint*** можна перейти до запису з відповідним номером.

Набори даних можуть знаходитися в відкритому і закритому стані. На це вказує властивість ***Active*** типу ***Boolean***. Якщо властивості ***Active*** встановлене значення ***True***, то НД відкритий. При встановленні властивості ***Active*** значення ***False*** набір даних знаходиться в закритому стані і зв'язок з базою даних розірваний. При зміні каталогів БД необхідно закривати і знову відкривати НД.

Керувати станом НД можна за допомогою методів ***Open*** і ***Close***. Процедура ***Open*** відкриває НД, її виклик еквівалентний встановленню властивості ***Active*** в значення ***True***. Під час виклику методу ***Open*** генеруються події ***BeforeOpen*** та ***AfterOpen***, а також викликаються процедури-обробники цих подій.

Процедура ***Close*** закриває НД, її виклик еквівалентний встановленню властивості ***Active*** в значення ***False***. Під час виклику методу ***Close*** генеруються події ***BeforeClose*** та ***AfterClose***, а також викликаються процедури-обробники цих подій. Перша з подій генерується безпосередньо перед відкриттям (закриттям), а інша після.

Наприклад, для заборони відкриття набору даних можна використати наступну процедуру:

procedure TForm1.Table1BeforeOpen(DataSet: TDataSet);
begin
if not CheckBox1.Checked then Abort;

end;

При закритті набору даних зміни не зберігаються, тому необхідно зберегти дані використовуючи метод ***Post***.

Кожне поле НД являє собою окремий стовпчик, для роботи з яким служить об'єкт ***Field*** типу ***TField***, та похідні по типах від нього об'єкти ***TIntegerField***, ***TStringField*** і т.д. Кількість полів набору даних визначається властивістю ***FieldCount***. Це число може відрізнятися від фізичного числа полів БД, тому що в набір даних не обов'язково включаються всі поля таблиці. Доступ до ***n***-го поля НД забезпечується властивістю ***Fields[Index:Integer]***. Оскільки в процесі роботи порядок полів може мінятися, то краще використовувати функції ***FindField (const FieldName : String):TField*** і ***FindByField***. Для доступу до значення поля поточного запису використовуються властивості об'єкту ***Field – AsInteger, AsString*** і т. п., які дозволяють звертатися до значення поля як до величини відповідного типу.

Для встановлення монопольного доступу до таблиці використовується властивість ***Exclusive***.

На етапі розробки додатка за допомоги **Редактора полів** можна створити для НД **статичні (стійкі) поля**, перевагами яких є:

1. Визначення обчислювальних полів, значення яких розраховуються за допомогою виразів, що використовують значення інших полів.
2. Обмеження складу полів НД.
3. Зміна порядку полів НД.
4. Сковування або показ окремих полів при виконанні додатку.
5. Визначення формату відображення або редагування даних поля на етапі розробки додатку.

При виконанні додатка можна визначити тип полів НД за допомогою властивості ***DefaultFields*** типу ***Boolean***. Якщо його значення дорівнює ***True***, то НД має поля за замовчуванням, тобто динамічні поля, в противному випадку для НД задані статичні поля.

Для запуску **Редактора полів** треба двічі клацнути на компоненті ***Table*** або ***Query*** або викликати для цих компонент контекстне меню і обрати пункт ***Fields Editor***. В заголовку **Редактора полів** виводиться складне ім'я НД. Більшу частину **Редактора полів** займає список статичних полів, при

чому поля перераховуються в порядку їх створення, який може відрізнятися від порядку полів в БД.(див. рис 3.1)

Спочатку список статичних полів порожній, вказуючи, що всі поля НД є динамічними. За допомогою **Редактора полів** розробник може:

- створити нове статичне поле;
- знищити статичне поле;
- змінити порядок слідування статичних полів.

Для створення статичних полів слід викликати контекстне меню **Редактора полів** і обрати пункт ***Add Fields***, в наслідок чого з'являється діалогове вікно додавання нових полів. В списку ***Available fields*** (доступні поля) вікна містяться всі ті поля НД, які ще не є статичними. Після вибору полів і натискання ***OK*** ці поля додаються в склад статичних полів НД. Статичне поле, яке добавлене, є полем даних і пов'язане з конкретним фізичним полем таблиці БД.

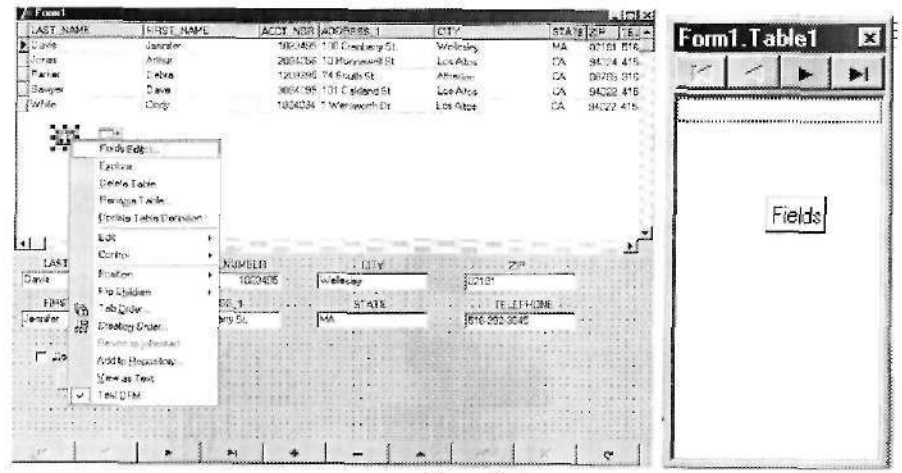


Рис 3.1

Щоби додати всі фізичні поля таблиці БД в контекстному меню **Редактора полів** обирається пункт ***Add all Fields***.

Для знищення статичного поля необхідно вибрати пункт ***Delete*** контекстного меню або натиснути кнопку <***Delete***>.

Порядок слідування полів визначається їх розташуванням в **Редакторі полів**. За замовченням порядок полів відповідає порядку фізичних в таблиці БД. Його можна змінювати, переміщаючись у списку за допомогою миші або комбінацією

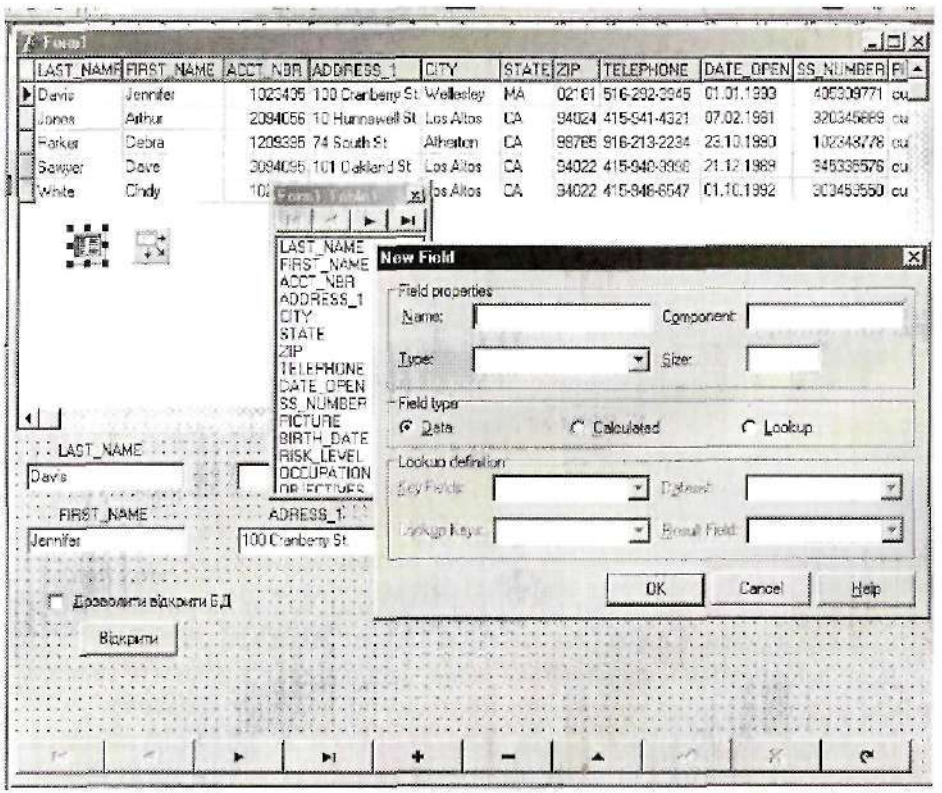
клавіш **<Ctrl>+<PageUp>** і **<Ctrl>+<PageDown>**. Існує три типа статичних полів:

1. Поле даних, яке пов'язане з відповідним фізичним полем таблиці.

2. Обчислювальне поле, значення якого розраховується в обробнику подій ***OnCalcFields*** в процесі виконання додатку.

3. Поле вибору, значення якого можна обирати зі списку, який формується на основі заданих критеріїв та правил.

Для створення нового статичного поля будь-якого типу треба обрати в контекстному меню **Редактора полів** пункт ***NewField***, в результаті чого відкривається діалогове вікно. Для завдання загальних властивостей (параметрів) нового поля використовується група елементів керування ***Field*** properties (властивості поля). В полі введення ***Name*** задається значення властивості ***Field Name*** (ім'я поля), а в полі введення



Component – значення властивості ***Name*** (ім'я компонента поля – об'єкт типу ***TField***). При програмуванні використовується ім'я поля. В списку ***Type*** і полі введення ***Size*** вказуються тип даних і розмір поля. Тип даних обов'язково задається для всіх полів, а необхідність завдання розміру залежить від типу даних.

Тип нового поля обирається в групі перемикачів ***Field type*** (див.рис.3.2) з наступних варіантів:

Data (поле даних);

Calculated (обчислювальне поле);

Lookup (поле вибору).

В групі ***Lookup definition*** (Визначення вибору) для поля вибору встановлюються такі параметри, як НД і поля зв'язку, а також поля для формування списку вибору і результату.

Після створення нового статичного поля його властивості стають доступними через **Інспектор об'єктів** і можуть бути змінені. При цьому кожному параметру, який задається за допомогою поля введення або перемикача вікна ***NewField***, відповідає відповідна властивість об'єкту типу ***TField*** (див. табл. 2). Всі властивості об'єкту доступні через **Інспектор об'єктів**, за винятком властивості ***FieldName***, яка доступна для читання тільки при виконанні додатку, але при розробці додатка значення цієї властивості видно у вікні **Редактора полів**. Значення параметра Type визначає клас об'єкту ***Field***, який буде відповідати статичному полю, наприклад для типу ***String*** – це ***TStringField***, а для типу ***Float*** – це ***TFloatField***.

Таблиця 2. Властивості об'єктів поля.

Властивість об'єкту поля	**Елемент керування**
Field Name	Поле введення Name
Name	Поле введення Component
Size (для рядкових полів)	Поле введення Size
Precision (для числових полів)	
FieldKind	Група перемикачів Field type
KeyFields	Комбінований список Key Fields
Lookup KeyFields	Комбінований список Lookup Keys

LookupDataset	Комбінований список Dataset
LookupResultField	Комбінований список Result Field

Властивість ***FieldKind*** типу ***TFieldKind*** визначає тип поля і приймає наступні значення:

fkData (поле даних);

fkCalculated (обчислювальне поле);

fkLookupField (поле вибору);

fkInternalCalc (обчислювальне поле, яке зберігається в НД);

fkAggregate (поле, яке містить агрегований результат).

Для створення **статичного обчислювального поля** треба виконати наступні дії:

1. У вікні створення статичного поля завдати ім'я і тип поля, а також вибрати перемикач ***Calculated***.

2. Для НД, який містить поле, підготувати код обробника події ***OnCalcFields*** типу ***TDataSetNotifyEvent***, в якому цьому полю присвоюється значення, яке вимагається, при цьому для розрахунку значення обчислювального поля можна використовувати значення інших полів, а також змінні та константи програми.

Після створення обчислювального поля його властивість ***FieldKind*** автоматично отримує значення ***fkCalculated*** Крім того, також властивість ***Calculated*** типу ***Boolean*** встановлюється в значення ***True***, яке вказує на те, що поле є обчислювальним. Для полів іншого типу властивість ***Calculated*** має значення ***False***.

Подія ***OnCalcFields*** призначена для визначення значень всіх обчислювальних полів НД. Вона генерується кожний раз при зчитуванні запису з таблиці, також при зміні необчислювальних полів, якщо властивість ***AutoCalcFields*** типу ***Boolean*** встановлена в значення ***True*** (за замовченням). На час виконання обробника події ***OnCalcFields*** НД переводиться в режим ***dsCalcFields*** розрахунку обчислювальних полів, а потім повертається в попередній режим.

Поле вибору дає можливість вибору одного значення із запропонованого списку і автоматичного занесення інформації в задане поле модифікованого запису. З полем вибору пов'язаний спеціальний список, який заповнюється значен-

ніями вказаного поля із другого НД. Обидва НД зв'язуються за полем (полями) зв'язку.

При зміні НД, за полем якого побудований список вибору, автоматичної зміни списку не виконується. Поновлення списку вибору є обов'язком програміста. Його зручно виконувати за допомогою методу ***RefreshLookupList***.

Операції з полями.

Через об'єкт типу ***TField*** розробник може:

1.Звернутися до поля та його значення.

2.Перевірити тип та значення поля.

3. Відформатувати значення поля, яке відображається та редагується в візуальних компонентах.

Ім'я статичного поля є складеним і за замовчуванням утворюється шляхом з'єднання імені НД і імені фізичного поля таблиці БД. Наприклад, якщо для фізичного поля ***Name*** НД ***Table1*** за допомогою **Редактора полів** створене статичне поле, то воно отримає ім'я ***Table1Name***.

Приклад звернення до поля:

Table1.FieldByName('Number').DisplayLabel1:='Кількість';

Table1Number. DisplayLabel1:='Кількість';

Тут для статичного поля ***Number*** можливі два способи звернення: за іменем поля в НД і за іменем об'єкта ***Field*** поля.

Для визначення порядкового номеру поля в НД можна використати властивість ***FieldNo*** типу ***Integer***, наприклад так:

```
var x: integer;
. . .
x:= Table1.FieldByName('Date').FieldNo;
```

Для доступу до значення поля служать властивості ***Value*** та ***AsXXX***. Властивість ***Value*** типу ***Variant*** – це фактичні дані в об'єкті типу ***TField***. При виконанні додатку ця властивість використовується для читання та запису значень в полі. Якщо програміст звертається до властивості ***Value***, то він має самостійно забезпечувати перетворення і узгодження типів значень полів і значень, які читаються або записуються.

Таблиця 3. Можливі типи властивості *Value*.

Тип об'єкта поля	Тип властивості Value
TField	Variant
TStringField, TBLOBField	String
TIntegerField, TSmallintField, TWordField, TAutoinc-	Longint

Field	
TBCDField, TFloatField, TCurrencyField	Double
TBooleanField	Boolean
TDateTimeField, TDate-Field, TTimeField	TDateTime

Оскільки при доступі до поля за допомогою властивості ***Value*** розробник має забезпечувати перетворення та узгодження типів значень, то частіше більш зручно використовувати варіанти властивості ***AsXXX***:

AsVariant типу ***Variant***;
AsString типу ***String***;
AsInteger типу ***Longint***;
AsFloat типу ***Double***;
AsCurrency типу ***Currency***;
AsBoolean типу ***Boolean***;
AsDateTime типу ***TDateTime***.

При використанні будь-яких з цих властивостей виконується автоматичне перетворення типу значення поля до типу, який відповідає назві властивості.

Перевірка типу та значення поля.

Для полів числових типів, напр, об'єктів типу ***TIntegerField*** і ***TFloatField*** властивості ***MinValue*** і ***MaxValue*** дозволяють встановити мінімальне та максимальне можливі значення, які можуть бути введені у відповідне поле. Тип цих властивостей визначається конкретним об'єктом типу ***TField.***

Table1Price.MinValue := 0;
Table1Price.MaxValue := 9999.99;
Table1Number.MinValue := 0;
Table1Number.MaxValue := 1000;

Обмеження на значення, які вводяться, можна визначити також за допомогою властивості ***CustomConstraint*** типу ***String.***

Table1Price.CustomConstraint := 'Price>=0 and Price<=9999.99';

Table1Number. CustomConstraint := 'Number>=0 and Number<=1000';

Розробник може не тільки обмежувати значення, що вводяться в поля, але і задавати через властивість ***DefaultExpression*** типу ***String*** значення за замовченням, яке автома-

тично заноситься в поле при додаванні нового запису. Це властивість рядкового типу, тому її значення береться в лапки:

Table1Number.DefaultExpression :='100';

Всі розглянуті події виникають до зміни значення поля, тому розробник може в необхідних випадках відмовитися від внесення зміни. Відмова від зміни значення поля для цих подій виконується різними способами. При використанні подій ***OnSetText*** типу ***TFieldSetTextEvent***, ***OnValidate*** і ***OnChange*** типу ***TFieldNotifyEvent*** розробник провокує виключення, в результаті чого записування значення в поле не виконується. Якщо виключення не генерується, то нове значення заноситься в поле. Так як обробник не отримує нового значення поля, для доступу до нього треба використати властивості ***Value*** або ***AsXXX.***

При редагуванні поля користувачем, події ***OnValidate*** та ***OnChange*** не викликаються до тих пір, поки не буде виконана спроба зберегти зміни, наприклад натисканням клавіші **<Enter>.**

Procedure TForm1.Table1NamberValidate (Sender: TField);
Begin
If (Table1.FieldByName('Code').AsString='1') or (Table1. FieldByName('Code').AsString='3') then Abort;
End;

Форматування відображеного значення поля.

Властивість ***DisplayFormat*** типу ***String*** керує відображенням значень полів в візуальних компонентах, наприклад ***DBGrid*** і ***DBEdit.*** Ця властивість діє для числових полів, а також для полів дати і часу. В якості значення властивості ***DisplayFormat*** задається маска, яка визначає формат відображення поля. Ця маска складається з трьох секцій, розділених символом «;». Секції задають форму відображення додатних, від'ємних та нульових значень. Якщо задана одна секція, то вона застосовується для виводу всіх значень.

Властивість ***EditFormat*** типу ***String*** задає маску, яку використовують при форматуванні значення числового поля перед його відображенням для редагування в візуальному компоненті. Формування цієї маски нічим не відрізняється від формування маски, що задається в ***DisplayFormat.***

Властивість ***EditMask*** типу ***String*** задає маску, що використовується при редагуванні значення, в візуальному ком-

понентi. Маска дозволяє обмежити число символів, які вводяться користувачем. Крім того, в інформацію, що вводиться, можна вставити додаткові символи. Для побудови маски зручно скористатися Редактором маски(подвійне клацання в полі значень властивості ***EditMask***).

Властивість ***DisplayWidth*** типу ***Integer*** визначає кількість символів, призначених для виводу значення поля в візуальному компоненті, який пов'язаний з цим полем. За замовчуванням значення цієї властивості визначається шириною фізичного поля таблиці, яка задана при створенні таблиці.

Заголовок поля визначається властивістю ***DisplayLable*** типу ***String***. За замовчуванням значенням цієї властивості є ім'я поля (***FieldName***), але, на відміну від останнього, властивість ***DisplayLable*** доступна і для запису, і її можна використовувати для настроювання інтерфейсу додатка.

```
Procedure TForm1.btnFilterClick (Sender: TObject);
Begin
Table1Name. DisplayLable := Edit.Text;
End;
```

Властивості сітки DBGrid.

За допомогою даного компонента можна управляти зовнішнім виглядом сітки, що відображає дані. Основною властивістю сітки є властивість ***Columns*** типу ***TDBGridColumns,*** яке являє собою масив об'єктів ***Column*** типу ***TColumn***, які описують окремі стовпчики сітки.

Властивість ***SelectedIndex*** типу ***Integer*** задає номер поточного стовпчика в масиві ***Columns***, а властивість ***SelectedField*** вказує на об'єкт типу ***TField***, якому відповідає поточний стовпчик сітки.

Властивість ***FieldCount*** типу ***Integer***, яка доступна при виконанні програми, містить кількість видимих стовпців сітки, а властивість ***Fields [Index: Integer]*** типу ***TField*** дозволяє отримати доступ до окремих стовпчиків. Індекс визначає номер стовпчика в масиві стовпців і приймає значення в інтервалі ***0..FieldCount-1.***

Властивості ***Color*** та ***FixedColor*** типу ***TColor*** задають колір сітки та її фіксованих елементів відповідно. За замовчуванням властивість ***Color*** має значення ***clWindow*** (колір фона Windows), а властивість ***FixedColor*** – значення ***clBtnFace*** (колір кнопки).

Властивість ***TitleFont*** типу ***TFont*** визначає шрифт, який використовується для виведення заголовків стовпців.

Доступ до параметрів сітки (напр., для настройки) можливий через властивість ***Options*** типу ***TGridOptions***. Ця властивість являє собою множину і приймає комбінації наступних значень:

dgEditing (користувачеві дозволяється редагування даних в комірках);

dgAlwaysShowEditor (сітка не блокує режим редагування);

dgTitles (відображаються заголовки стовпчиків);

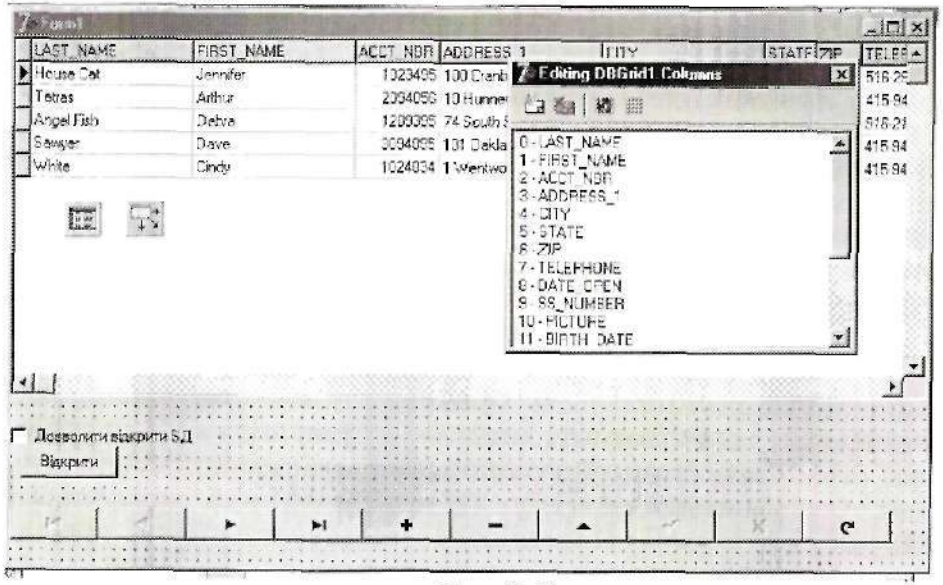


Рис. 3.3

dgIndicator (для поточного запису в початку рядку виводиться вказівник);

dgColumnResize (користувач може за допомогою миші змінювати розмір стовпців та пересувати їх);

dgColLins (між стовпцями виводяться розділяючі вертикальні лінії);

dgRowLines (між рядками виводяться розділяючі горизонтальні лінії);;

dgTsbs (для переміщення по сітці можна використовувати клавіші ***<Tab>*** та ***<Shift>+<Tab>***);

dgRowSelect (користувач може виділити цілий рядок); при встановленні цього значення ігноруються значення dgEditing і dgAlwaysShowEditor;

dgAlwaysShowSelection (комірка залишається виділеною, навіть якщо сітка тіряє фокус);

dgConfirmDelete (при видаленні рядка задається запит на підтвердження операції);

dgCancelOnExit (додані до сітки порожні рядки при загублені сіткою фокуса не зберігаються в НД);

dgMultiSelect (в сітці можливо одночасно виділити декілька рядків).

За замовчуванням властивість Options містить комбінацію значень [***Options, dgTitles, dgTitles, dgColumnResize, dgColLins, dgRowLines, dgTsbs, dgTsbs, dgCancelOnExit***].

При клацанні на комірці з даними генерується подия ***OnCellClick***, а клацання на заголовку стовпчику викликає подію ***OnTitleClick***. Обидві події мають тип ***TDBGrigClickEvent***, який описується так:

type TDBGrigClickEvent = procedure (Column: TColumn) of object;

Параметр ***Column*** – це стовпчик, на якому було здійснене клацання. Якщо властивість ***Options*** містить значення ***dgColumnResize***, то користувач може за допомогою миші пересувати стовпчики сітки. При такому переміщенні генерується подія ***OnColumnMoved*** типу ***TMovedEvent***, яка описується так:

type TMovedEvent = procedure (Sender: TObject; From Index, ToIndex: Longint) of object;

Параметри ***From Index*** і ***ToIndex*** вказують індекси в масиві стовпчиків сітки, які відповідають попередньому та новому положенню переміщеного стовпчика відповідно.

Для програмної реалізації відображення сітки використовується обробник події ***OnDrowColumnCell*** типу ***TDrowColumnCellEvent***, яке виникає при промальовуванні будь-якої комірки. Тип події описується так:

type TDrowColumnCellEvent = procedure (Sender: TObject; const Rect: TRect; DataCol: Integer; Column: TColumn; State: TGridDrawState) of object;

Параметр ***Rect*** містить координати прямокутника, який обмежує комірку прямокутника, параметр ***DataCol*** визначає номер промальованого стовпчика в масиві стовпчиків сітки, а параметр ***Column*** є об'єктом стовпчика, який промальовується. Параметр ***State*** задає стан комірки і приймає комбінації наступних значень:

gdSelected (комірка знаходиться у вибраному диапазоні);

gdFocused (комірка має фокус введеня);

gdFixed (комірка знаходиться в фіксованому діапазоні).

Порядок виклику події ***OnDrowColumnCell*** залежить від значення властивості ***DefaultDrawing*** типу ***Boolean***. Якщо властивість має значення ***True*** (за замовчуванням), то перед тим, як генерується подія ***OnDrowColumnCell*** в комірці відображається фон і виводиться інформація. Потім навколо обраної комірки малюється прямокутник вибору. Якщо властивість ***DefaultDrawing*** має значення ***False***, то одразу викликається подія ***OnDrowColumnCell***, в обробнику якої слід розмістити операції по промальовуванню області сітки.

Окремий стовпчик ***Column*** сітки являє собою об'єкт типу ***TColumn***. За замовчуванням для кожного поля НД, пов'язаного з компонентом ***DBGrid***, автоматично створюється окремий стовпчик, і всі стовпчики в сітці є доступними. Такі стовпчики є ***динамічними***. Для створення ***статичних*** стовпчиків використовується спеціальний Редактор стовпців (Р.с.). Якщо хоча б один стовпчик сітки є статистичним, то динамічні стовпчики вже не створюються ні для одного із полів НД, що залишилися. При чому в НД доступними є статистичні стовпці, а решта стовпчиків вважаються відсутніми. Змінити склад статистичних полів можна за допомогою Р.с. на етапі розробки додатка.

Характеристики і поведінка сітки та її окремих стовпчиків в більшості визначаються полями НД, для яких створюються об'єкти типу ***TColumn***. Об'єкти типу ***TColumn*** статичних стовпчиків створюються на етапі розробки додатка, і їх властивості доступні через **Інспектор об'єктів**.

Для запуску **Редактора стовпців** (Р.с.) можна викликати контекстне меню компонента ***DBGrid*** і вибрати в ньому пункт

Columns Editor. Р.с. можна викликати також клацанням миші на властивості ***Columns*** у вікні **Інспектора обєктів**. Під заголовком знаходиться панель інструментів, видимістю якої можна керувати за допомогою пункту ***Toolbar*** контекстного меню Р.с. (див.рис. 3.3)

Спочатку список статичних стовпців пустий, тим самим демонструючи, що всі стовпці сітки є динамічними. **Редактор стовпців** дозволяє:

- створити статичний стовпець;
- видалити статичний стовпець;
- змінити порядок слідування статичних стовпчиків.

Крім того, для будь-якого обраного в Р.с. стовпчика (об'єкту типу ***TColumn***) через **Інспектор об'єктів** можна завдати або змінити його властивості та визначити обробники його подій. Це допустимо тому, що відповідні статичним стовпчикам об'єкти типу ***TColumn*** доступні вже на етапі виконання програми.

Статичний стовпчик можна завдати наступними способами:

- натискуванням кнопки панелі інструментів Р.с.;
- вибором пункту ***Add*** контекстного меню Р.с.;
- натисканням клавіші ***<Insert>***.

В будь-якому випадку до списку додається рядок, який відповідає новому статичному стовпчику. В лівії частині рядка міститься номер цього стовпчика в масиві стовпчиків, а в правій – ім'я поля НД, з яким пов'язаний стовпчик. Одразу після додававння до списку стовпчик не пов'язаний ні з одним полем і замість імені поля вказується ***TColumn***. Під час виконання додатка подібний стовпчик виявиться порожнім. Що би зв'язати стовпчик із будь-яким полем, треба встановити значення його властивості ***FieldName***.

Для додавання в список статичних стовпчиків, які відповідають всім полям НД, треба натиснути кнопку панелі інструментів Р.с. або вибрати в його контекстному меню пункт ***Add All Fields.***

Для видалення зі списку статичних стовпців необхідно їх виділити, після чого виконати одну з наступних дій:

- натиснути кнопку панелі інструментівв Р.с.;
- обрати пункт ***Delete*** контекстного меню Р.с.;
- натиснути клавішу ***<Delete>***.

Знову створені статичні стовпці дістають значення властивостей за замовчуванням, які залежать також від полів НД, з якими ці стовпці пов'язані. Якщо значення властивостей були змінені через **Інспектор об'єктів** і вимагається відновити їх початкові значення, то це можливо виконати натискуванням кнопки або вибором пункту ***Restore Defaults*** (Відновити параметри за замовчуванням) контекстного меню Р.с..

Об'єкт стовпця доступний через властивість ***Columns*** типу ***TDBGridColumns***. При проектуванні додатка властивості цього об'єкту (тобто стовпця, обраного в списку Р.с.) доступні через **Інспектор об'єктів**.

Властивості об'єкта стовпця:

Alignment типу ***TAlignment*** - керує вирівнюванням значень в комірках стовпця і може приймати наступні значення:

taLeftJustify (вирівнювання по лівій границі);

taCenter (вирівнювання по центру);

taRightJustify (вирівнювання по правій границі);

Count типу ***Integer*** – вказує кількість стовпців сітки;

Field типу ***TField*** – визначає об'єкт поля НД, який пов'язаний із стовпцем;

FieldName типу ***String*** – вказує ім'я НД, з яким пов'язаний стовпець. При встановленні цієї властивості за допомогою **Інспектора об'єктів** значення можна обирати в списку;

PickList типу ***TString*** – список для вибору значень, які заносяться в поле. Поточна комірка разом зі списком ***PickList*** утворює свого роду компонент ***ComboBox*** або ***DBComboBox.*** Якщо для стовпця сформований список вибору, то при спробі редагування комірки цього стовпчика справа зя'вляється стрілка, при натискуванні якої список розкривається і дозволяє обрати одне із значень. При цьому можна ввести в комірку будь-яке допустиме значення;

Title типу ***TColumnTitle*** – об'єкт заголовка стовпця. В свою чергу цей об'єкт має такі властивості, як ***Caption, Alignment, Color, Font***, що визначають назву, вирівнювання, колір, шрифт заголовка відповідно.

Властивості стовпця, які керують форматуванням, видимістю або можливістю модифікації значень, не відрізняються від відповідних властивостей поля.

DBGrid1. Columns[1].Visible := False;

Контрольні завдання:

Після засвоєння матеріалу лабораторної роботи слід виконати наступні задачі:

1. Порахувати кількість входжень букви «о» в дані другого стовпчика таблиці та замінити знайдені букви символом «?».
2. Організувати можливість заборони відкриття чи закриття набору даних використовуючи події ***BeforeOpen*** та ***BeforeClose***; вивести повідомлення про заборону.
3. Вміти створити статичне поле.
4. Вміти створити обчислювальне поле.
5. Використовуючи властивості ***Table*** заборонити редагування набору даних, задати обмеження на значення полів, заховати довільний стовпець.
6. Створити статичні стовпці.
7. Вміти задати список для вибору значень.
8. Вміти задати властивості відображених даних (колір, тип шрифту і т. д.).

9.Оформити короткий звіт про виконану роботу, внести в нього основні теоретичні відомості.

Лабораторна робота №4

Тема: Навігаційний спосіб доступу до даних. Пошук, сортування та фільтрація даних.

Мета роботи: вивчити навігаційні можливості середовища **Delphi**, вивчити основні властивості та методи, що пов'язані з організацією пошуку та фільтрації в наборі даних.

Хід роботи.

Навігаційний спосіб доступу до даних полягає в обробці кожного окремого запису НД. Цей спосіб доступу дає можливість виконувати наступні операції:

1. Сортування записів.
2. Навігацію по НД.
3. Редагування записів.
4. Вставку та видалення записів.
5. Фільтрацію записів.

Крім дій з окремими записами, за допомогою компонента ***Table*** можна також виконувати **дії з таблицею БД** в цілому, наприклад, створювати, знищувати, перейменовувати, встановлювати режими доступу до таблиць.

Для створення таблиці використовують метод ***CreateTable.*** В результаті на диску з'являється порожня таблиця. Перед викликом методу треба підготувати необхідні дані, на основі яких створюється таблиця. Ці дані треба присвоїти у якості значень відповідним властивостям НД. Перед викликом методу ***CreateTable*** НД має бути закритий і встановлені значення наступних властивостей:

DatabaseName (шлях до файлів БД (каталог));
TableType (тип таблиці);
FieldDefs (описання полів);
IndexDefs (описання індексів);
TableName (назва таблиці).

```
procedure TForm1.ButtonNewTableClick(Sender:TObject);
begin
//Закриття НД
Table1.Active:=False;
Table1 .DatabaseName:='BDPlace';
Table1. TableName:='New Table';
Table1. TableType:=ttParadox;
//Описання полів
```

```
Table1. FieldDefs.Clear;
Table1. FieldDefs.Add('Code',ftAutoinc,0,true);
Table1. FieldDefs.Add('Name',ftString,20,true);
Table1. FieldDefs.Add('Date',ftDate,0,false);
//Описання індексів
Table1. IndexDefs.Clear;
Table1. IndexDefs.Add(' ','Code',[ixPrimary,ixUnique]);
Table1. IndexDefs.Add('indName','Name',[ixCaseInsensitive]);
//Створення таблиці
Table1.CreateTable;
//Встановлення поточного індексу
Table1.IndexName:='indName';
//Відкриття НД, пов'язаного з новою пустою таблицею.
Table1.Active:=true;
end;
```

Сортування набору даних виконується автоматично за поточним індексом. При зміні індексу виконується автоматичне перевпорядкування записів. Таким чином сортування можливе за полями, для яких створено індекс. Напрямок сортування визначає параметр ***ixDescending*** поточного індексу, за замовченням він включений і сортування виконується в порядку зростання значень. Задати індекс, за яким виконується сортування, можна за допомогою властивостей ***IndexName*** або ***IndexFieldNames***. Сортування за ключем можна тільки за допомогою властивості ***IndexFieldNames.***

```
Procedure TForm1.Button4Click(Sender:TObject);
Begin
Case RadioGroup1.ItemIndex of
0: Table1. IndexName:='indName';
1: Table1. IndexName:='indBirthDay';
2: Table1. IndexFieldNames :='Code';
End;
```

Навігація по набору даних полягає в керуванні покажчиком поточного запису. Для переміщення покажчика поточного запису використовують наступні методи:

процедура ***First*** – встановлення на перший запис;
процедура ***Last*** – встановлення на останній запис;
процедура ***Prior*** – встановлення на попередній запис;
процедура ***Next*** – встановлення на наступний запис;

процедура ***FindFirst*** типу ***Boolean*** – намагається встановити курсор на перший запис НД і повертає ***True*** у випадку успіху;

процедура ***FindNext*** - намагається встановити курсор на наступний запис НД і повертає ***True*** у випадку успіху;

процедура ***FindLast*** - намагається встановити курсор на останній запис НД і повертає ***True*** у випадку успіху;

процедура ***FindPrior*** - намагається встановити курсор на попередній запис НД і повертає ***True*** у випадку успіху;

функція ***MoveBy(Distance:Integer):Integer*** – переміщення на число записів, що визначається параметром ***Distance***. Якщо його значення додатне, то переміщення виконується вперед, а якщо від'ємне, то назад. Переміщення виконується по записам НД, які він містить в поточний момент часу. Число записів визначається властивістю ***RecordCount.***

```
procedure TForm1.Button3Click(Sender:TObject);
var s: real; n: integer;
begin
s:=0;
Table1.First;
for n:=1 to Table1. RecordCount do begin
s:= s+Table1.FieldByName('Salary').AsFloat;
Table1.Next;
end;
Label2.Caption:=FloatToStr(s);
end;
```

Для контролю положення покажчика поточного запису можна використовувати властивість ***RecNo.*** Для таблиць Paradox властивість ***RecNo*** можна використовувати ще і для переходу до запису із відомим номером: такий перехід виконується встановленням властивості ***RecNo*** в значення, яке рівне номеру потрібного запису.

```
Procedure TForm1.Button1Click(Sender:TObject);
Begin
Table1.RecNo:=StrToInt(Edit1.Text);
End;
```

Для визначення початку та кінця НД при переміщенні покажчика поточного запису використовують властивості ***BOF*** та ***EOF*** типу ***Boolean*** відповідно.

Для виконання дій з записами можна використовувати всі типи циклів. Нехай початковим буде перший запис набору даних.

```
with Table1 do begin
First;
while NOT EOF do
begin { Набір операторів }
Next ;
end;
end;

with Table1 do begin
First;
repeat { Набір операторів }
until NOT FindNext;
end;
```

Аналогічно можна організувати цикл від кінця до початку.

```
with Table1 do begin
Last;
while NOT BOF do
begin { Набір операторів }
Prior;
end;
end;

with Table1 do begin
Last;
repeat { Набір операторів }
until NOT FindPrior;
end;
```

Якщо набір даних НД дозволяє звертання до запису по номеру (його властивість ***IsSequenced*** в даному випадку повинно бути рівне ***True***), можна використовувати цикл ***for***. Наприклад

```
if Books.IsSequenced then
for RecNo := 1 to Books.RecordCount do
begin { Набір операторів }
end;
```

Важливо відмітити, що якщо в тілі циклу виконувати редагування БД, то зміщення поточного запису може привести

до непередбачуваних ситуацій. Наприклад наступна процедура:

```
with Table1 do while not EOF do begin
Delete;
Next;
end;
```

не призведе до знищення всіх записів БД, тому що і оператор ***Delete*** і ***Next*** переміщають запис вперед, що в результаті призведе до пропускання певних записів. Правильною буде конструкція без оператора ***Next***.

Закладки використовують в тому випадку, коли необхідно тимчасово залишити поточний запис, але потім знову повернутися до нього. Для цієї мети набор даних володіє наступними методами: ***GetBookmark*** (створює закладку), ***GotoBookmark*** (переміщує курсор на закладку), ***FreeBookmark*** (знищує закладки), ***BookmarkValid*** (перевіряє закладку), ***CompareBookmark*** (порівнює дві закладки).

```
var
MyBookmark: TBookMark;   // Опис
...
begin
MyBookmark := Table1.GetBookmark;   // Створення
if Table1.BookmarkValid(MyBookmark) then
Table1.GotoBookmark(MyBookmark); // Перехід
...
if Table1.BookmarkValid(MyBookmark) then
Table1.FreeBookmark(MyBookmark);  // Вивільнення ресурсів
...
```

Пошук записів, що відповідають деякій умові. Метод *Locate*.

Метод ***Locate*** шукає перший запис, який задовольняє умові і робить його поточним. В даному випадку у якості результату процедури буде значення ***True***. Якщо пошук не дав результатів, то повертається значення ***False***, а курсор (покажчик на поточний запис) не змінює свого положення. Структура методу має вигляд:

function Locate(const *KeyFields:* ***String; const*** *KeyValues:* ***Variant;***
Options: ***TLocateOptions): Boolean;***

Список ***KeyFields*** вказує на поле чи список полів, по яким ведеться пошук. У випадку декількох полів їхні назви розділяються крапкою з комою. В свою чергу, критерії пошуку задаються в варіантному масиві ***KeyValues*** таким чином, що *j*-ве значення в ***KeyValues*** ставиться у відповідність *j*-вому полю у ***KeyFields***. За допомогою параметру ***Options*** можна вказати необов'язкові параметри пошуку:

- *loCaseInsensitive* – пошук ведеться без врахування регістру;
- *loPartialKey* – запис буде задовольняти умові пошуку, якщо він містить частину шуканого контексту.

Метод *Lookup*.

Метод ***Lookup*** знаходить запис, який задовольняє умові пошуку, але не робить його поточним, а повертає значення деяких його полів. Незалежно від результату пошуку запису покажчик поточного запису в НД не змінюється. На відміну від ***Locate***, метод ***Lookup*** виконує пошук тільки на точну відповідність критерію пошуку значення поля пошуку запису.

function *Lookup(**const** KeyFields:* ***String; const*** *KeyValues:* ***Variant;***

const *ResultFields:* ***String): Variant;***

Список ***KeyFields*** вказує на поле чи список полів, по яким ведеться пошук. У випадку декількох полів їхні назви розділяються крапкою з комою. Параметр ***KeyValues*** визначає пошукові значення полів, список яких міститься в параметрі ***KeyFields.*** В параметрі ***ResultFields*** перечислюються всі поля, значення яких потрібно отримати, при умові що пошук дасть позитивний результат. Тип результату – ***Variant*** або варіантний масив. Якщо пошук ведеться по декількох полях, їх необхідно приводити до типу варіантного масиву за допомогою функції перетворення ***VarArrayOf.***

Якщо в результаті пошуку запис не знайдений, то процедура повертає ***Null***. Це можна перевірити за допомогою наступного прийому:

```
var
LookupResult: Variant;
begin
LookupResult := Table1.Lookup(...);
if VarType(LookupResult) = varNull then . . .
. . .
```


У випадку позитивного завершення пошуку, ***Lookup*** повертає із знайденого запису значення тих полів, список яких містить ***ResultFields***.

Фільтрація записів.

Для завдання виразу фільтра використовується властивість ***Filter*** типу ***String***. Фільтр – це конструкція, в склад якої можуть входити наступні елементи:

- імена полів таблиць;
- літерали;
- операції порівняння;
- арифметичні операції;
- логічні операції;
- круглі та квадратні дужки.

Літерал – це значення, яке задане явно, наприклад, число, рядок або символ. Імена змінних у виразі фільтра використовувати не можна. Якщо в фільтр треба включити значення змінної або властивості якої-небудь компоненти, то це значення має бути перетворене в рядковий тип.

В якості логічних операцій можна використовувати *AND, OR, NOT*. Круглі дужки застосовуються для зміни порядку виконання арифметичних та логічних операцій. Якщо вираз фільтра не дозволяє сформувати складний критерій фільтрації, то в доповнення до нього можна використовувати обробник події ***OnFilterRecord.***

Для *активації* та *деактивації* фільтра застосовується властивість ***Filtered*** типу ***Boolean.*** За замовчуванням ця властивість має значення ***False***, і фільтрація виключена. При встановленні ***Filtered*** в значення ***True*** фільтрація включається і в НД відбираються записи, які задовольняють фільтру, який записаний у властивість ***Filter***. Якщо вираз фільтра не заданий (за замовчуванням), то в НД попадають всі записи.

Параметри фільтрації задаються за допомогою властивості ***FilterOptions*** типу ***TFilterOptions***. Ця властивість належить до множинного типу і може приймати комбінації двох значень:

- *foCaseInsensitive* – регістр букв не враховується;
- *foNoPartialCompare* – виконується перевірка на повну відповідність вмісту поля і значення, заданого для пошуку. Застосовується для рядків символів. Якщо відомі тільки перші символи (або символ) рядка, то треба вказати їх у виразі філь-

тра, замінивши решта символів на зірочки (*) і виключивши значення *foNoPartialCompare*.

За замовчуванням всі параметри фільтра включені, і властивість ***FilterOptions*** має значення [].

При включенні фільтрації шляхом встановлення властивості ***Filtered*** в значення ***True*** для кожного запису, що відбирається в НД генерується подія ***OnFilterRecord*** типу ***TFilterRecordEvent.*** В процесі відбору записів НД автоматично переводиться в режим ***dsFilter,*** при цьому в ньому забороняються дії, пов'язані із зміною записів. Тип ***TFilterRecordEvent*** описується наступним чином:

type TFilterRecordEvrnet = procedure (DataSet: *TDataSet;* ***var Accept:*** *Boolean)* ***of Object;***

Логічний параметр ***Accept*** вказує, чи включати у НД, що визначається параметром ***DataSet***, запис, для якого згенерована подія ***OnFilterRecord.*** За замовчуванням параметр ***Accept*** має значення ***True***, і в НД включаються всі записи, що задовольняють умові фільтра, який визначається властивістю ***Filter***. При встановленні параметра ***Accept*** в значення ***False*** запис в НД не включається.

В обробнику події ***OnFilterRecord*** можна визначати додаткові до виразу фільтра умови фільтрації. За своєю дією основні і додаткові умови з'єднані логічною операцією ***AND,*** тобто для відбору запису в НД вимагається виконання обох умов. На відміну від виразу фільтру, в обробнику події ***OnFilterRecord*** можна кодувати будь-які складні перевірки за допомогою засобів мови **Delphi.**

При ***фільтрації по діапазону*** в НД включаються записи, значення полів яких попадають в заданий діапазон, тобто умовою фільтрації є вираз вигляду *значення > нижньої границі* ***AND*** *значення < верхньої границі* (замість операторів порівняння < і > можна використовувати оператори <= і >=). Така фільтрація застосовується до НД ***Table.*** Цей спосіб фільтрації застосовується тільки для індексованих полів. Індекс поля, діапазон якого заданий у якості критерію для відбору записів, має бути встановлений як поточний за допомогою властивості ***IndexName*** або ***IndexFieldNames.*** Якщо поточний індекс не встановлений, то за замовчуванням використовується головний індекс.

Для *включення* та *виключення* по діапазону застосовуються методи ***ApplyRange*** і ***CancelRange.*** Перший з них активує фільтр, а другий – деактивує. Попередньо для індексного поля (полів), за яким виконується фільтрація, слід завдати діапазон допустимих значень.

Методи ***SetRangeStart*** і ***SetRangeEnd*** встановлюють нижню та верхню границі діапазону відповідно. Названі процедури не мають параметрів, і для завдання границь діапазону використовується просто інструкція присвоювання. При цьому методи ***SetRangeStart*** і ***SetRangeEnd*** переводять НД в режим ***dsSetKey***.

Для зміни попередньо встановлених границь діапазону призначені методи ***EditRangeStart*** і ***EditRangeEnd,*** дія яких аналогічна дії методів ***SetRangeStart*** і ***SetRangeEnd*** відповідно.

Сумісно з цими методами використовується властивість ***KeyExclusuve*** типу ***Boolean***, яке визначає, як враховується задане граничне значення при аналізі записів. Якщо властивість ***KeyExclusuve*** має значення ***False*** (за замовчуванням), то записи, у яких записи полів фільтрації співпадають з границями діапазону, включаються в склад НД, якщо ж властивість має значення ***True,*** то такі записи в НД не попадають. Властивість ***KeyExclusuve*** діє окремо для нижньої та верхньої границі. Значення цієї властивості має встановлюватися одразу після виклику методів ***EditRangeStart, EditRangeEnd, SetRangeStart*** і ***SetRangeEnd.***

Розглянемо Приклад, в якому керування фільтрацією НД виконується за допомогою двох кнопок і поля редагування. При натисканні кнопки **Фільтрувати** (***btnFilter***) фільтр активізується шляхом присвоювання значення ***True*** властивості ***Filtered*** НД. Редактор ***edtFilter*** призначений для завдання виразу фільтра. При активізації фільтра виконується відбір записів, які задовольняють умові, що задана у виразі. При натисканні кнопки **Всі записи** (***btnAllRecord***) фільтр відключається, при цьому показуються всі записи.

```
Procedure TForm1.FormCreate (Sender: TObject);
Begin
Table1.FilterOptions: [foCaseInsensitive];
Table1.Filtered:= False;
End;
```

```
Procedure TForm.btnFilterClick (Sender: TObject);
Begin
Table1.Filtered:=True;
Table1.Filter:=edtFilter.Text;
End;

Procedure TForm.btnAllRecordClick (Sender: TObject);
Begin
Table1.Filtered:=False;
End;
```

При включенні фільтрації шляхом встановлення властивості ***Filtered*** в значення ***True*** для кожного запису, що відбирається в НД генерується подія ***OnFilterRecord*** типу ***TFilterRecordEvent***. В процесі відбору записів НД автоматично переводиться в режим ***dsFilter,*** при цьому в ньому забороняються дії, пов'язані із зміною записів. Тип ***TFilterRecordEvent*** описаний наступним чином:

type TFilterRecordEvent = procedure (DataSet: *TDataSet;* ***var Accept:*** *Boolean)* ***of Object;***

Логічний параметр ***Accept*** вказує, чи включати в НД., що визначається параметром ***DataSet***, запис, для якого згенерована подія ***OnFilterRecord.*** За замовчуванням параметр ***Accept*** має значення ***True,*** і в НД включаються всі записи, що задовольняють умові фільтра, який визначається властивістю ***Filter.*** При встановленні параметра ***Accept*** в значення ***False*** запис в НД не включається. На відміну від виразу фільтра, в обробнику події ***OnFilterRecord*** можна кодувати будь-які складні перевірки за допомогою засобів мови **Delphi**.

Т. ч., НД ***Table*** допускає два способу завдання умов фільтрації: за допомогою виразу фільтра ***Filter*** та в обробнику події ***OnFilterRecord.***

Зв'язування таблиць.

Між окремими таблицями БД може існувати зв'язок, який організується через поля зв'язку таблиць. Поля зв'язку обов'язково мають бути індексованими. Зв'язок між таблицями визначає відношення один до багатьох (***Master- Detail***). Після встановлення зв'язку між таблицями при переміщенні в головній таблиці курсору на будь-який запис, в підлеглій таблиці автоматично стають доступними записи, у яких значення поля зв'язку дорівнює значенню поля зв'язку поточного запису головної таблиці.

Для організації зв'язку між таблицями в підлеглій таблиці використовуються наступні властивості, які вказують:

MasterSource – джерело даних головної таблиці;

IndexName – поточний індекс підлеглої таблиці;

IndexFieldNames – поле або поля зв'язку поточного індексу підлеглої таблиці;

MasterFields – поле або поля зв'язку підлеглої таблиці.

Як правило таблиці зв'язуються через **Інспектор об'єктів**. При цьому для встановлення властивостей ***IndexName*** і ***MasterFields*** зручно використовувати спеціальний **Редактор полів зв'язку** (***Field Link Designer***), який викликається подвійним клацанням в області значення властивості ***MasterFields*** у вікні **Інспектора об'єктів**. В списку ***Available Indexes*** (*Доступні поля*) обирається індекс підлеглої таблиці, після чого поля, які його складають, відображаються в списку ***Detail Fields*** (*Детальні поля*). В цьому списку необхідно обрати поле підлеглої таблиці, а в списку ***MasterFields*** (*Головне поле)* – поле головної таблиці. Після натискання ***Add*** обрані поля пов'язуються між собою, що відображається в списку ***Joined Fields*** (*Зв'язані поля*). Заповнення властивостей ***IndexName*** і ***MasterFields*** виконується після закриття вікна натисканням ОК.

Контрольні завдання:

Після засвоєння матеріалу лабораторної роботи слід вміти:

1. Організувати навігацію по набору даних різними способами.
2. Здійснювати сортування записів.
3. Здійснювати фільтрацію НД.
4. Здійснювати пошук заданого значення поля згідно певного критерію.
5. Створити програму, яка дає можливість здійснювати навігацію по НД за допомогою створених програмістом кнопок (використовуючи методи ***First, Last, Next, Prior, MoveBy***),створити закладки та організувати можливість переходу на задану закладку, сортувати записи обраного поля, фільтрувати набір даних, вводячи критерій, здійснювати пошук в обраному полі згідно введеного з клавіатури значення (ко-

```
Procedure TForm.btnFilterClick (Sender: TObject);
Begin
Table1.Filtered:=True;
Table1.Filter:=edtFilter.Text;
End;

Procedure TForm.btnAllRecordClick (Sender: TObject);
Begin
Table1.Filtered:=False;
End;
```

При включенні фільтрації шляхом встановлення властивості ***Filtered*** в значення ***True*** для кожного запису, що відбирається в НД генерується подія ***OnFilterRecord*** типу ***TFilterRecordEvent***. В процесі відбору записів НД автоматично переводиться в режим ***dsFilter,*** при цьому в ньому забороняються дії, пов'язані із зміною записів. Тип ***TFilterRecordEvent*** описаний наступним чином:

type TFilterRecordEvent = procedure (DataSet: *TDataSet;* ***var Accept:*** *Boolean)* ***of Object;***

Логічний параметр ***Accept*** вказує, чи включати в НД., що визначається параметром ***DataSet***, запис, для якого згенерована подія ***OnFilterRecord.*** За замовчуванням параметр ***Accept*** має значення ***True,*** і в НД включаються всі записи, що задовольняють умові фільтра, який визначається властивістю ***Filter.*** При встановленні параметра ***Accept*** в значення ***False*** запис в НД не включається. На відміну від виразу фільтра, в обробнику події ***OnFilterRecord*** можна кодувати будь-які складні перевірки за допомогою засобів мови **Delphi.**

Т. ч., НД ***Table*** допускає два способу завдання умов фільтрації: за допомогою виразу фільтра ***Filter*** та в обробнику події ***OnFilterRecord.***

Зв'язування таблиць.

Між окремими таблицями БД може існувати зв'язок, який організується через поля зв'язку таблиць. Поля зв'язку обов'язково мають бути індексованими. Зв'язок між таблицями визначає відношення один до багатьох (***Master- Detail***). Після встановлення зв'язку між таблицями при переміщенні в головній таблиці курсору на будь-який запис, в підлеглій таблиці автоматично стають доступними записи, у яких значення поля зв'язку дорівнює значенню поля зв'язку поточного запису головної таблиці.

Для організації зв'язку між таблицями в підлеглій таблиці використовуються наступні властивості, які вказують:

MasterSource – джерело даних головної таблиці;

IndexName – поточний індекс підлеглої таблиці;

IndexFieldNames – поле або поля зв'язку поточного індексу підлеглої таблиці;

MasterFields – поле або поля зв'язку підлеглої таблиці.

Як правило таблиці зв'язуються через **Інспектор об'єктів**. При цьому для встановлення властивостей ***IndexName*** і ***MasterFields*** зручно використовувати спеціальний **Редактор полів зв'язку** (***Field Link Designer***), який викликається подвійним клацанням в області значення властивості ***MasterFields*** у вікні **Інспектора об'єктів**. В списку ***Available Indexes*** (*Доступні поля*) обирається індекс підлеглої таблиці, після чого поля, які його складають, відображаються в списку ***Detail Fields*** (*Детальні поля*). В цьому списку необхідно обрати поле підлеглої таблиці, а в списку ***MasterFields*** (*Головне поле)* – поле головної таблиці. Після натискання ***Add*** обрані поля пов'язуються між собою, що відображається в списку ***Joined Fields*** (*Зв'язані поля*). Заповнення властивостей ***IndexName*** і ***MasterFields*** виконується після закриття вікна натисканням ОК.

Контрольні завдання:

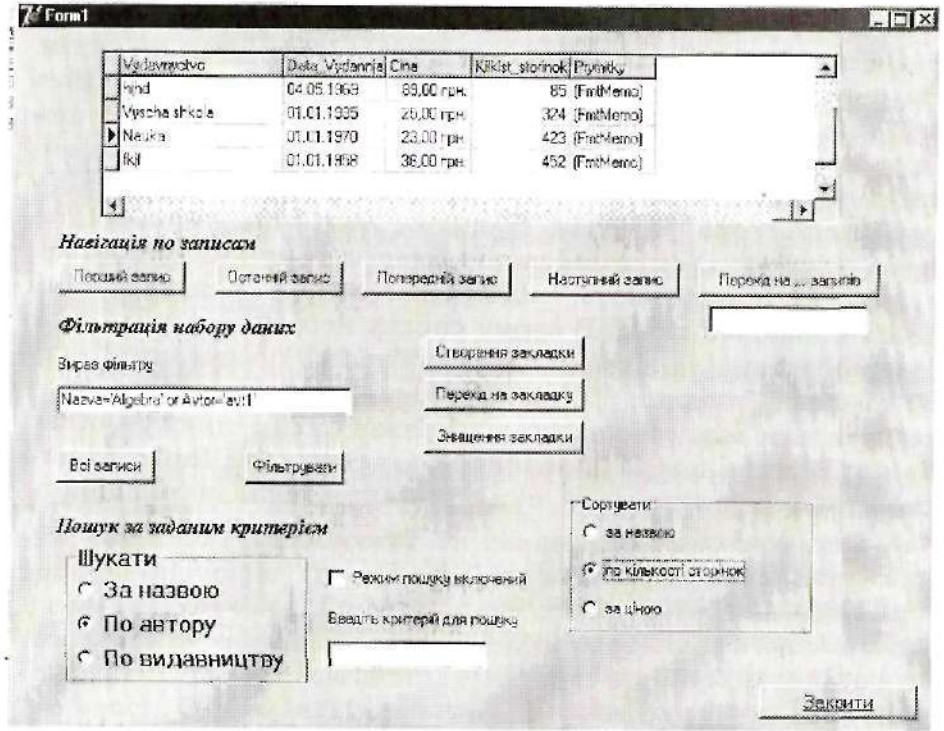
Після засвоєння матеріалу лабораторної роботи слід вміти:

1. Організувати навігацію по набору даних різними способами.
2. Здійснювати сортування записів.
3. Здійснювати фільтрацію НД.
4. Здійснювати пошук заданого значення поля згідно певного критерію.
5. Створити програму, яка дає можливість здійснювати навігацію по НД за допомогою створених програмістом кнопок (використовуючи методи ***First, Last, Next, Prior, MoveBy***),створити закладки та організувати можливість переходу на задану закладку, сортувати записи обраного поля, фільтрувати набір даних, вводячи критерій, здійснювати пошук в обраному полі згідно введеного з клавіатури значення (ко-

ристуючись методами ***Locate*** та ***Lookup***). Після створення додатку слід отримати наступний вигляд форми додатку.

6. В середовищі програмування **Delphi** створити таблицю, яка містить 5 полів і 7 записів. За полями зв'язку дану таблицю зв'язати з двома підлеглими для неї таблицями.

7. Оформити короткий звіт про виконану роботу, внести в нього основні теоретичні відомості.



Лабораторна робота №5

Тема: Набір даних *Query*. Основні поняття мови *SQL*. Організація *SQL*-запитів.

Мета роботи: вивчити реляційні можливості середовища **Delphi**, вивчити основні властивості та методи, що пов'язані з організацією пошуку та фільтрації в наборі даних ***Query***, вміти створювати ***SQL***-запити з простими та складними умовами відбору.

Хід роботи.

Особливості набору даних Query.

Компонент ***Query*** являє собою НД, записи якого формуються в результаті виконання ***SQL***-запитів і базуються на реляційному способі доступу до даних. Текст запиту на основі якого в НД відбираються записи міститься у властивості ***SQL*** типу ***TString***. Запит включає в себе команди на мові ***SQL*** і виконується при відкритті набору даних. При формуванні запиту на стапі розробки можна використовувати текстовий редактор, який викликається подвійним клацанням в області значення властивості ***SQL***.(див. рис. 5.1)

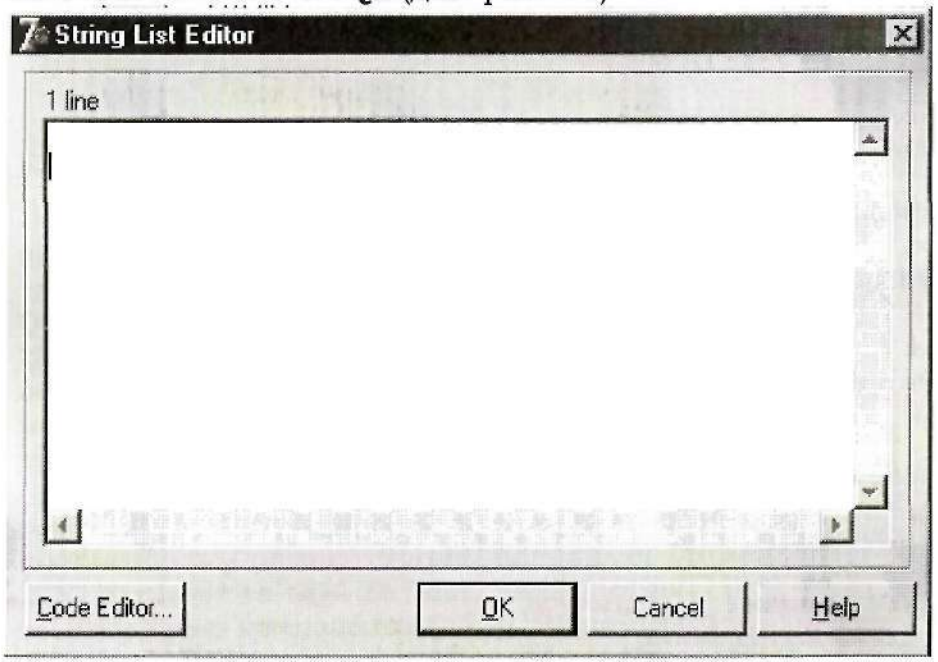


Рис. 5.1

Запит також можна змінювати динамічно, вносячи в його текст зміни безпосередньо під час виконання додатку.

Створимо найпростіший додаток, який дозволяє будувати та виконувати ***SQL***-запити. Він має наступну структуру:

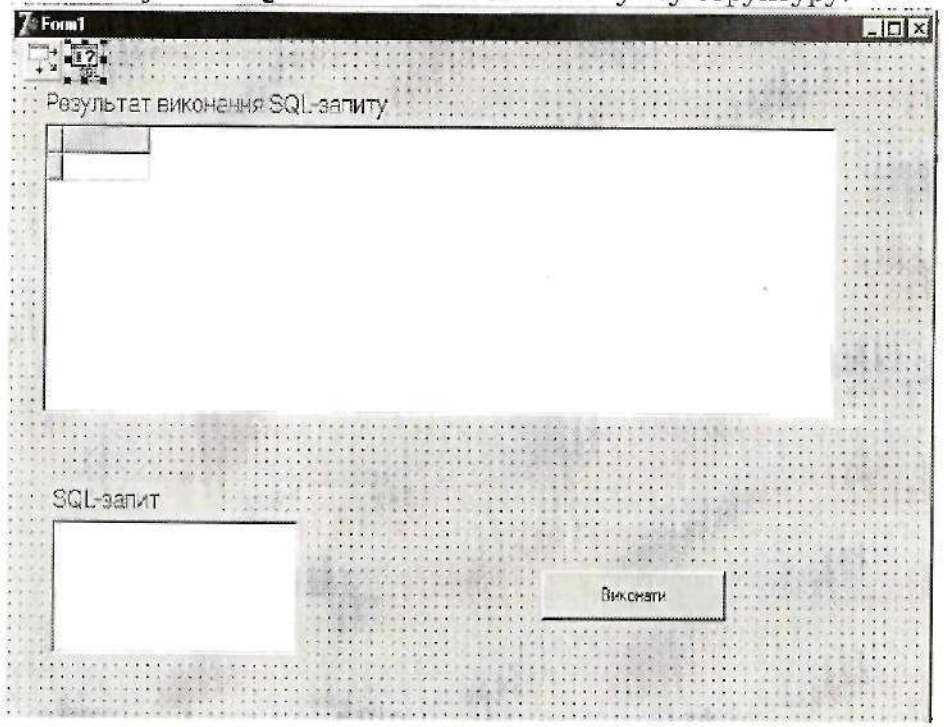


Рис 5.2

Відповідно код модуля вказаної форми буде наступним:

```
interface
uses
Windows, Messages, SysUtils, Variants, Classes, Graphics,
Controls, Forms,
Dialogs, StdCtrls, Grids, DBGrids, DB, DBTables;
type
TForm1 = class(TForm)
Query1: TQuery;
Memo1: TMemo;
DataSource1: TDataSource;
DBGrid1: TDBGrid;
Button1: TButton;
```



```
Label1: TLabel;
Label2: TLabel;
procedure FormCreate(Sender: TObject);
procedure Button1Click(Sender: TObject);
private
{ Private declarations }
public
{ Public declarations }
end;
var
Form1: TForm1;
implementation
{$R *.dfm}

procedure TForm1.FormCreate(Sender: TObject);
begin
DataSource1.DataSet:=Query1;
DBGrid1.DataSource:=DataSource1;
end;

procedure TForm1.Button1Click(Sender: TObject);
begin
Query1.Close;
Query1.SQL.Assign(Memo1.Lines);
Query1.Open;
end;
end.
```

Метод ***Assign*** виконує присвоєння одного об'єкт іншому, при цьому об'єкти повинні мати сумісні типи. Формування результуючого набору виконується при активізації компонента ***Query*** викликом методу ***Open*** або установкою властивості ***Active*** значення ***True.*** В деяких випадках нема необхідності повертати набір даних, наприклад при видаленні, вставці або модифікації записів, тоді слід виконувати запит викликом методу ***ExecSQL.***

Крім того, набрати та виконати в інтерактивному режимі текст ***SQL***-запиту дозволяють інструментальні програми, які поставляються разом з **Delphi**, наприклад ***Database Desktop, SQL Explorer*** та ***SQL Builder***, який викликається через контекстне меню компонента ***Query.*** Якщо необхідна можливість редагування записів, то властивості ***RequestLive*** потрібно

встановити значення ***True.*** Щоб перевірити результат встановлення значення властивості ***RequestLive*** можна скористатися властивістю ***CanModify*** типу ***Boolean***. Якщо вона має значення ***True,*** то НД є таким, який допускає редагування, якщо ***False*** – не допускає редагування.

Основні поняття мови SQL.

Мова ***SQL*** надає для використання наступні функції:

- статистичні:
  - AVG (середнє значення);
  - MAX (максимальне значення);
  - MIN (мінімальне значення);
  - SUM (сума);
  - COUNT (кількість значень);
  - COUNT (*) (кількість ненульових значень);
- функції роботи з рядками:
  - UPPER (*Str*) (перетворення символів рядка *Str* до верхнього регістру);
  - LOWER (*Str*) (перетворення символів рядка *Str* до нижнього регістру);
  - TRIM (*Str*) (знищення пропусків на початку та в кінці рядка *Str*);
  - SUBSTRING (*Str* FROM *n1* TO *n2*) (виділення із рядка *Str* підрядка, який включає в себе символи, починаючи з позиції *n1* і закінчуючи позицією *n2*);
  - CAST (<*Expression*> AS <*Type*>) (зведення виразу *Expression* до типу *Type*);
- функції декодування дати та часу:
  - EXTRACT (<*Елемент*> FROM <*Вираз*>) (із виразу, який містить значення дати або часу, забирається значення, що відповідає вказаному елементу).

Визначення даних – це маніпулювання цілими таблицями. Сюди включаються наступні операції:

- створення нової таблиці;
- знищення таблиці;
- зміна складу полів таблиці;
- створення та знищення індексу.

Для **створення таблиці** використовується інструкція ***CREATE TABLE,*** яка має наступний формат: ***CREATE TABLE*** *<Ім'я таблиці >*

(<Ім'я поля> <Тип поля >,

...

<Ім'я поля> <Тип поля>);

Типи даних мови ***SQL*** та відповідні їм типи для таблиці **Paradox** наведені в таблиці 4. Тут N означає довжину поля в байтах, X – число цифр, Y – число цифр після десяткової точки.

Таблиця 4. Типи даних мови *SQL* та відповідні їм типи для таблиці Paradox.

SQL	Paradox
SMALLINT	SHORT
INTEGER	LONG INTEGER
DECIMAL (X, Y)	BCD
NUMERIC (X, Y)	NUMBER
FLOAT (X, Y)	FLOAT (X, Y)
CHARACTER(N)	ALPHA
VARCHAR (N)	ALPHA
DATE	DATE
BOOLEAN	LOGICAL
BLOB (N, 1)	MEMO
BLOB (N, 2)	BINARY
BLOB (N, 3)	FORMATTED MEMO
BLOB (N, 4)	OLE
BLOB (N, 5)	GRAPHIC
TIME	TIME
TIMESTAMP	TIMESTAMP
MONEY	MONEY
AUTOINC	AUTOINCREMENT
BYTES (N)	BYTES

Для таблиці **Paradox** можна визначити ключ, вказав команду PRIMARY KEY і перерахувавши в дужках після нього поля, що утворюють цей ключ. Ключові поля в списку полів мають бути першими.

Приклад

CREATE TABLE Personnel1.db
(Code AUTOINC,
Name CHAR (20),
Position CHAR (15),
Salary NUMERIC (10, 2),
PRIMARY KEY (Code);

Для **видалення таблиці** призначена інструкція:

DROP TABLE <Ім'я таблиці>;

Зміна складу полів таблиці полягає в додаванні або знищенні полів, що веде до зміни її структури. Ця операція виконується інструкцією:

ALTER TABLE <Ім'я таблиці>
ADD <Ім'я поля> <Тип поля>,
DROP <Ім'я поля>,
...
ADD <Ім'я поля> <Тип поля>,
DROP <Ім'я поля>;

Операнд ***ADD*** додає до таблиці нове поле, а операнд ***DROP*** знищує з таблиці поле із заданим ім'ям.

Приклад

ALTER TABLE Personnel1.db
ADD Section SMALLINT,
ADD Note CHAR(30),
DROP Position;

Індекс створеннюється інструкцією:

CREATE INDEX <Ім'я індексу> ON <Ім'я таблиці> (<Ім'я поля1>, . . . , <Ім'я поляN>);

Однією інструкцією можна створити один індекс, при цьому одне поле може входити в склад декількох індексів. При сортуванні за індексом записи впорядковуються в порядку зростання значень індексних полів.

Приклад

CREATE INDEX
indNamePosition ON Personnel.db (Name, Position)

Для **видалення індексу** призначена інструкція:

DROP INDEX <Ім'я таблиці> . <Ім'я індексу>;

Для ілюстрації особливостей створення ***SQL***-запитів зручно використовувати утиліту ***SQL Explorer***. Після виклику утиліти необхідно розшукати у вікні браузера раніше створений псевдонім, клацніть на значку вузла зліва від нього, щоб дістати доступ до потрібної БД. Оскільки ця БД – файл-серверна (або, швидше за все, локальна), в ній зберігаються тільки таблиці. Після розкриття вузла ***Tables*** і клацання на імені будь-якої таблиці, треба перейти на вкладку ***Enter SQL.***(Див. рис.5.3)

Вибір даних з таблиць полягає в отриманні з них полів і записів, які задовольняють заданим умовам. Результат виконання запиту, на основі якого обираються записи, називають ***вибіркою***. Дані можна обирати із однієї або декількох таблиць за допомогою інструкції ***SELECT***, яка має наступний формат:

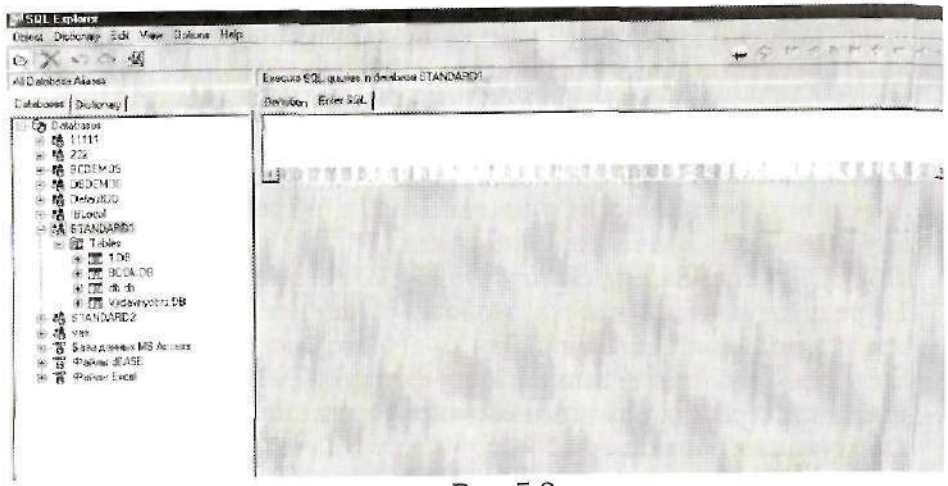


Рис. 5.3

SELECT [DISTINCT] {|< Список_полів>} FROM < Список_таблиць>*
[WHERE<Умова вибору>]
[ORDER BY <Список полів для сортування>]
[GROUP BY<Список полів для групування>]
[HAVING <Умови групування>]
[UNION< Вкладена інструкція SELECT >]

Список полів має містити хоча б одне поле. Якщо в НД треба включити всі поля таблиці, то замість перерахування імен полів можна вказати символ '*'. При явному вказані імен полів ми можемо керувати порядком їх слідування в НД. Оператор ***DISTINCT*** дозволяє або забороняє повторення записів в результуючому НД. Якщо він відсутній, то в НД дозволено повторення записів (тобто ці записи мають однакові значення полів). Операнд ***WHERE*** задає умови відбору, яким мають задовольняти записи в результуючому НД. Операнд ***ORDER BY*** містить список полів, які визначають порядок сортування записів результуючого НД. Операнд ***GROUP BY*** дозволяє виділити групи записів в результуючому НД. Групою називають записи з однаковими значеннями в полях, перерахованих за операндом ***GROUP BY.*** Операнд ***HAVING*** діє сумісно з операндом ***GROUP BY*** і використовується для вибору записів у самих групах.

Інструкції ***SELECT*** можуть мати складну структуру і бути вкладені одна в одну. Для об'єднання інструкцій використовують операнд ***UNION***, в якому розміщується вкладена інструкція

SELECT. Результуючий НД – це записи, які відібрано із врахуванням виконання умов вибору, заданих операндами ***WHERE*** обох інструкцій.

Крім фізичних полів таблиці, в НД можна включати ***обчислювальні*** поля. Для отримання обчислювального поля в списку полів вказується не ім'я цього поля, а вираз, за яким розраховується його значення. Наприклад:

*SELECT Name, Salary, Salary*1.5 FROM Personnel1.db*

В результуючому наборі можна об'єднати дані, які містяться в декількох таблицях. Для цього після слова ***FROM*** переховують імена таблиць, із записів яких формується НД. Таке використання даних із різних таблиць називають з'єднанням.

Сортування записів.

Для сортування використовується секція, що починається зарезервованими словами ORDER BY (сортувати по):

Приклад

SELECT Name, Position, Salary
FROM Personnel1.db
ORDER BY Position, Salary DESC

Як і при встановленні зв'язку між таблицями, сортувати можна по будь-яких полях, не обов'язково індексних. За замовченням сортування виконується в порядку зростання значень полів. Якщо вимагається задати для поля сортування в порядку спадання, то після імені цього поля вказується інструкція ***DESC.***

Простий критерій вибору.

Критерій вибору – логічний вираз, в якому можна використовувати операції, перераховані нижче.

1.Порівняння. Можливі оператори :

- = (дорівнює);
- > (більше)
- < (менше)
- >= (більше або дорівнює)
- <= (менше або дорівнює)
- <> або != (не дорівнює)
- !> (не більше)
- !< (не менше)

Приклад

SELECT Name FROM Personnel1.db
WHERE Salary>=4000

2.***LIKE*** (порівняння за шаблоном). У виразах операції ***LIKE*** допускається використання шаблону, в якому дозволені всі алфавітно-цифрові символи (із врахуванням регістру). Два символи мають спеціальне призначення:

% - заміна будь-якої кількості символів, в тому числі і нульового;

_- заміна одного символу.

Приклад

SELECT Name FROM Personnel1.db
WHERE Name LIKE "Ab"

3.***IS NULL*** (перевірка на нульове значення), яка має формат

<ВИРАЗ> IS [NOT] NULL

Приклад

*SELECT *FROM Store*
WHERE S_Price IS NULL

4.***IN*** (перевірка на входження значення у список), яка має формат

<ВИРАЗ> [NOT] IN <Список значень>

Приклад

SELECT Name, Salary FROM Personnel1.db
WHERE LOWER(Post) IN ("асистент","лаборант")

5. ***BETWEEN*** (перевірка на входження значення в діапазон). Її формат

<ВИРАЗ> [NOT] BETWEEN

Приклад

*SELECT *FROM Cards*
WHERE C_Date BETWEEN "21.5.09" AND "27.6.09"

Складний критерій відбору складається з:

- простих умов;
- логічних операцій:
  - ***AND*** (логічне І);
  - ***OR*** (логічне АБО);
  - ***NOT*** (логічне НІ);
- круглих дужок.

За допомогою секції ***WHERE*** можна не тільки зв'язувати таблиці, але і створювати досить складні критерії відбору да-

них. Для цього в секції указується довільна кількість операторів вигляду ПОЛЕ ~ЗНАЧЕННЯ, які пов'язані логічними операціями ***NOT, AND, OR***, де знак (~) означає операцію відношення (ці операції ***SQL*** співпадають з операціями відношення **Delphi**):

SELECT Cards. FROM Story, Cards*

WHERE (S.Quantity>250) AND (C.Date BETWEEN "21.12.10" AND "31.12.10")

ORDER BY NaklID

В даному прикладі відбираються усі записи про рух товару, кількість якого на складі перевищує 250 одиниць, на протязі заданого періоду. Звернемо увагу на дві обставини. По-перше, пріоритет операцій відношення в ***SQL*** вище за пріоритет логічних операцій – це позбавляє від необхідності розстановки численних дужок. По-друге, операція ***OR*** має менший пріоритет, ніж операція ***AND***.

Модифікація записів.

Модифікація записів полягає в редагуванні записів, вставленні в НД та знищення з НД записів.

Редагування записів – зміна значень полів в групі записів. Воно виконується інструкцією наступного формату:

UPDATE <ІМ'Я ТАБЛИЦІ>

SET <ІМ'Я ПОЛЯ1> = <ВИРАЗ1>

. . .

<ІМ'Я ПОЛЯ N> = <ВИРАЗ N>

[WHERE <УМОВИ ВИБОРУ >];

Приклад

UPDATE Personnel1 SET Salary = Salary +200

WHERE Salary < 1500

В одній інструкції ***UPDATE*** можна змінювати значення декількох полів. В цьому випадку для кожного з них вказується відповідне значення. Якщо оператор ***WHERE*** не заданий, то змінюються значення усіх вказаних полів.

*UPDATE Store SET Price=Price*1.28*

Вставка записів виконується за допомогою інструкції ***INSERT***, яка дозволяє додавати до таблиці одну або декілька записів. Для одного запису:

INSERT INTO <ІМ'Я ТАБЛИЦІ>

[(<СПИСОК ПОЛІВ>)]

VALUES (<СПИСОК ЗНАЧЕНЬ>);

В результаті виконання цієї інструкції до таблиці, ім'я якої вказане після слова ***INTO***, додається один запис. Для добавленого запису заповнюються поля, перераховані в списку. Значення полів беруться зі списку, розміщеного після слова ***VALUES.***

Приклад

INSERT INTO Store (Name, Price,Quantity)

VALUES ("Monitor", 1999, 10);

При додаванні до таблиці декількох значень інструкція ***INSERT*** має формат:

INSERT INTO <ІМ'Я ТАБЛИЦІ>

(<СПИСОК ПОЛІВ>)

ІНСТРУКЦІЯ SELECT;

Значення полів нових записів визначаються через значення полів записів, відібраних за допомогою інструкції ***SELECT***. Кількість добавлених записів дорівнює кількості відібраних записів. Список значень полів, що повертається інструкцією SELECT, має відповідати списку інструкцій ***INSERT*** за кількістю і типу полів.

Приклад

INSERT INTO CardArchives (Code, Move, Date)

SELECT C.Code, C.Move, C.Date

FROM Cards

WHERE C.Date BETWEEN 21.12.10 AND 31.12.10

Знищення групи записів виконується за допомогою інструкції ***DELETE***:

DELETE FROM <ІМ'Я ТАБЛИЦІ>

[WHERE <УМОВИ ВИБОРУ>];

Якщо критерій вибору не заданий, то знищуються всі записи.

Для налагодження ***SQL***-запиту при виконанні додатка в його тексті можна використовувати **параметри**. Параметр – це спеціальна змінна, перед ім'ям якої ставиться **":"** в тексті запиту. Цей знак не є частиною імені параметра і ставиться тільки в тексті запиту. В процесі виконання додатка замість параметра надається його значення. Параметри зручно використовувати для передачі в текст ***SQL***-запиту зовнішніх значень.

Приклад.

```
procedure TForm1.Button1Click(Sender: TObject);
begin
Query1.Close;
Query1.SQL.Clear;
Query1.SQL.Add('SELECT*');
```

```
Query1.SQL.Add('FROM BOOK.db');
Query1.SQL.Add(WHERE Kilkisty<='+Edit1.Text);
Query1.Open;
end;
```

Цю задачу можна розв'язати через включення в текст запиту параметра *prmKilkisty*

```
SELECT * FROM BOOK.db
WHERE Kilkisty >=: prmKilkisty;
```

Система **Delphi** автоматично враховує всі вказані в ***SQL***-запиті параметри в спеціальному списку параметрів, які є для НД ***Query*** значенням властивості ***Params*** типу ***TParams***, який є масивом. Щоби звернутися до параметра при виконанні додатка, слід вказати його номер (індекс) в списку параметрів: ***Params[1]***.

На стапі розробки додатка можна викликати Редактор параметрів в Інспекторі об'єктів. Тип кожного параметра треба вказати у властивості ***DataType*** типу ***TFieldType***.

```
procedure TForm1.Button1Click(Sender: TObject);
begin
Query1.Close;
Query1.ParamByName('prmKilkisty').AsFloat:=StrToFloat(Edit1.T
ext);
Query1.Open;
end;
```

Контрольні завдання:

Після засвоєння матеріалу лабораторної роботи слід вміти:

1.Створити найпростіший додаток, який дозволяє виконувати та створювати ***SQL***-запити.

2.Створити наступні види ***SQL***-запитів:

a)простий запит на вибірку;
b)запит на вибірку зі зміною порядку слідування полів;
c)запит зі складним критерієм;
d)запит з унікальними значеннями;
e)запит з використанням обчислювального поля;
f) запит на створення (видалення) таблиці;
g)запит на створення (видалення) індексів;
h) запит із сортування по заданому полю;
i) запит на зміну складу полів;
j) запит з використанням дій по модифікації записів;
k)запит з використанням параметра.

3. Оформити короткий звіт про виконану роботу, внести в нього основні теоретичні відомості.

Лабораторна робота №6

Тема: Створення клієнт-серверної бази даних InterBase.

Мета роботи: навчитися створювати клієнт-серверні БД, освоїти основні поняття архітектури «клієнт-сервер»,

Хід роботи.

В мережевій архітектурі клієнт-сервер БД розміщена на комп'ютері - сервері мережі і називається віддаленою БД. Програма, що виконує роботу з цією БД, розміщена на комп'ютері користувача. Програма користувача є **клієнтом**, її також називають **додатком-клієнтом**. Взаємодія їх відбувається наступним чином: клієнт формує і відсилає запит (***SQL***-запит) серверу, на якому розміщена БД. Сервер виконує запит і видає клієнту в якості результатів необхідні дані. Після обробки, інформація зберігається на сервері.

При роботі в режимі клієнт-сервер програма користувача повинна:

- виконувати з'єднання і відключення від БД;
- на основі запитів, отримувати від сервера необхідну інформацію;
- виконувати обробку даних (обробка даних принципово не відрізняється в порівнянні з локальними базами даних).

Віддалена БД являє собою сукупність взаємозв'язаних таблиць. Але ці таблиці, як правило, містяться в одному загальному файлі.

Зауваження. Для реалізації **реляційного** способу доступу до віддаленої БД за допомогою BDE необхідно використовувати тільки засоби мови ***SQL.*** Крім того, для НД не можна використовувати методи, властиві навігаційному способу доступу, наприклад, ***NEXT*** та ***PREVIOUS*** для переміщення поточного покажчика або ***EDIT, INSERT, APPEND*** або ***DELETE*** для зміни записів. *Для віддалених БД поле називається стовпчиком.*

Для організації БД використаємо сервер ***InterBase***. Вся інформація про БД даного сервера знаходиться в одному файлі з розширенням ***gdb***.

Типи даних в ***InterBase*** приведені в таблиці 5:

Таблиця 5. Типи даних в *InterBase*

Тип	Опис	Діапазон	
SMALLINT	Ціле число	-32768	32767
INTEGER	Ціле число	-2147483648	2147483647
FLOAT	Число з плаваючою крапкою	$3,4*10^{-38}$	$3,4*10^{38}$
DOUDLE PRECISION	Число з плаваючою крапкою	$1,7*10^{-308}$	$1,7*10^{308}$
CHARACTER(N) VARCHAR(N)	Рядкова змінна довжиною N	1	32767
DATE	Дата	01.01.0100	11.12.5941
BLOB	Двійкові дані довільного типу		

Логічний тип замінює CHAR(1).

Для виконання будь-яких операцій з БД з нею необхідно виконати з'єднання, тобто відкрити БД. Після закінчення роботи з'єднання потрібно розірвати, або закрити БД. Для з'єднання з БД програми типа ***IBConsole*** мають відповідні засоби, які викликаються за допомогою вікна меню. З'єднання з віддаленою БД виконаємо за допомогою програми ***IBConsole*** і компонентів Delphi, які підтримують механізм доступу BDE. Перед встановленням з'єднання з БД із програми ***IBConsole*** (**Пуск/Програми/*InterBase***) треба за допомогою команд ***Server/ Register*** виконати реєстрацію сервера ***InterBase***, а також завдати ім'я та пароль користувача. На початку на сервері ***InterBase*** для користувачів є стандартне ім'я ***SYSDBA*** та пароль «***masterkey***».

Для створення віддаленої БД ***InterBase*** знову використаємо програму ***IBConsole*** і команду ***Server/ Register***. У наступному вікні:

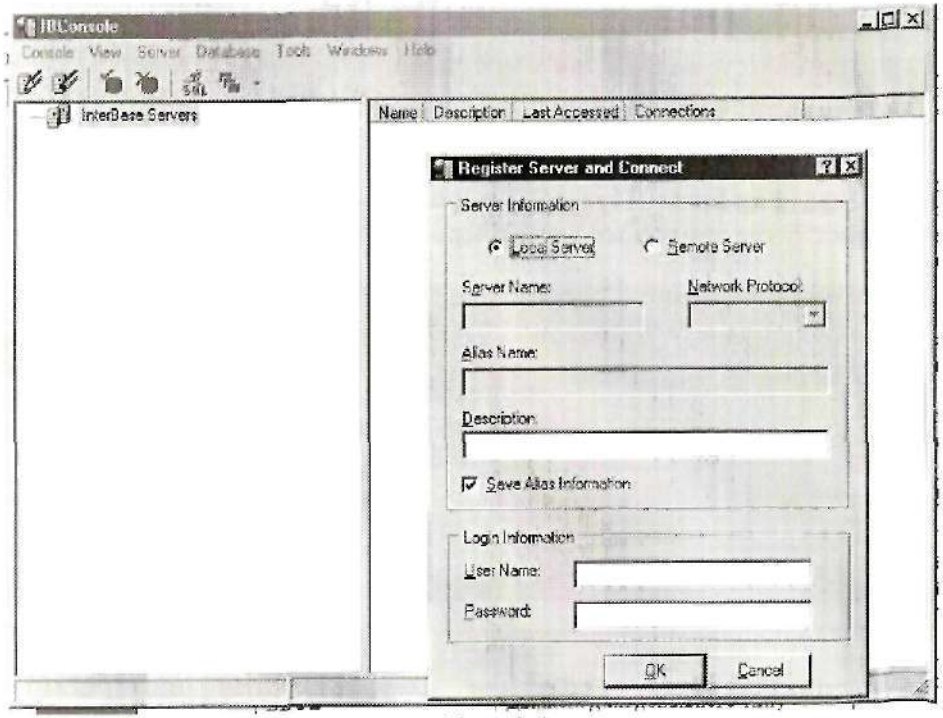


Рис 6.1

задаємо параметри:

Server Information – Local Server

User Name – SYSDBA

Password – masterkey.

В дереві ***Local Server*** вибираємо ***Databases*** і в його контекстному меню обираємо ***Create Database***...

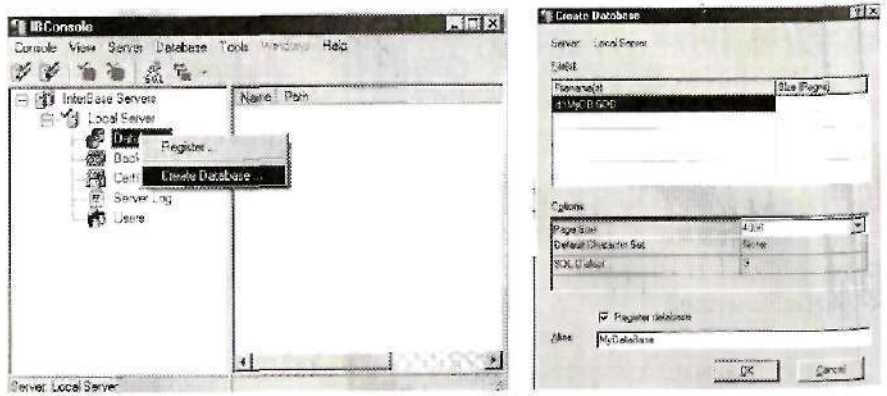


Рис 6.2

В полі ***Filenames*** вказуємо повний шлях до БД, далі задаємо назву аліасу, а інші параметри залишаємо без змін. Після натискання на кнопку "***OK***" буде створена БД. Із новою створеною БД автоматично встановлюється з'єднання.

Для створення таблиць, індексів і т. п. використасмо програму ***SQL Explorer*** (**Пуск/Програми/*Borland Delphi***). У вікні ***Object*** її головного меню вибирасмо пункт ***New***...

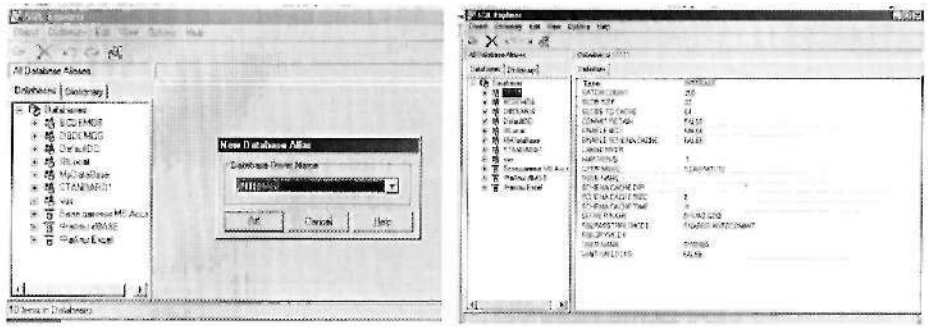


Рис 6.3

У списку поля ***Database Driver Name*** вибираємо ***INTERBASE***. У відповідних параметрах робимо наступні зміни. Полю ***USERNAME*** присвоюємо значення ***SYSDBA***, в полі ***SERVER NAME*** вказуємо повний шлях до бази. При подвійному натисканні лівої кнопки миші на назві аліасу з'явиться вікно ***DataBase Login***, де потрібно визначити ***User Name – SYSDBA***, ***Password – masterkey***. Після цього відкриється дерево алісу з вузлами ***Domains, Tables, View, Stored Procedures...***Для створення нової таблиці, натиснемо правою кнопкою миші на вузлі ***Tables*** і в контекстному меню виберемо ***New(Ctrl+N).*** Після створення нової таблиці одержимо можливість редагувати вузли ***Columns, Indices, Primary Key***...(відмітимо, що у полі ***Text*** бачимо текст ***SQL***- запиту)

Відповідним чином створюється потрібна кількість полів та задаються необхідні індекси. Для збереження змін натискаємо ***Apply (Ctrl+A)*** в контекстному меню.

Для з'єднання з БД із додатку треба створити необхідну форму. Нехай в формі розміщені сітка ***DBGrid1,*** джерело даних ***DataSourse1*** і набір даних ***Query1***. Різниця між формою для віддаленої БД і формою для локальної БД полягає в тому, що додатково організується з'єднання з БД. Для цього необхідно:

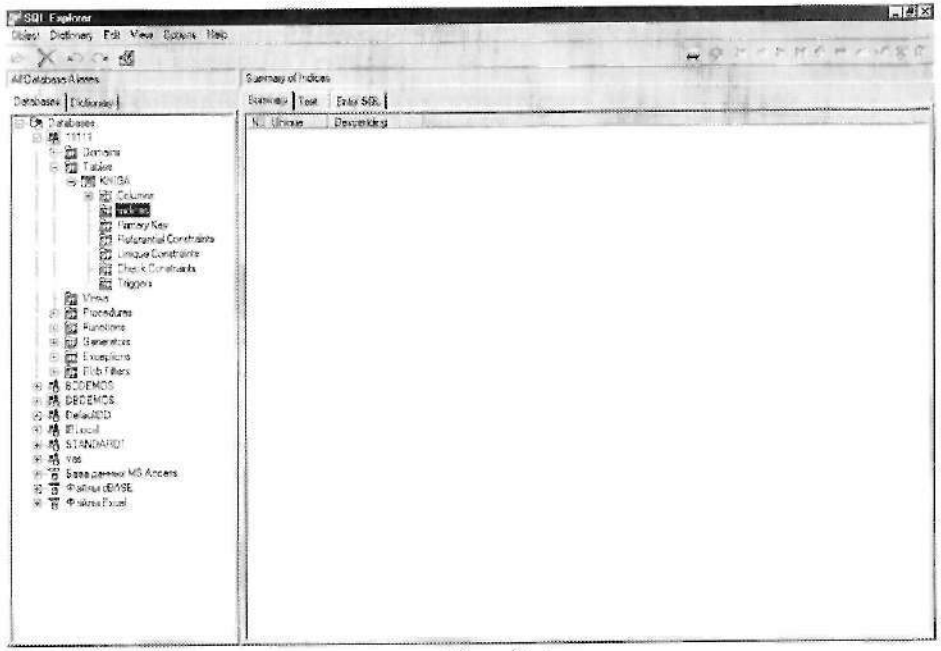


Рис 6.4

1.Створити псевдонім БД.

2.Розмістити в формі компонент ***Database***, встановити потрібні значення його властивостей і зв'язати з НД.

Для з'єднання з БД властивостям ***AliasName*** та ***DatabaseName*** компоненти ***Database1*** надають відповідно значення – псевдонім і ім'я БД. Це ім'я не співпадає ні з псевдонімом ні з власним ім'ям БД (ім'ям головного її файлу). Воно діє тільки в додатку і тільки для організації підключення до БД, вказаної її псевдонімом. З'єднання з БД встановлюється наступними способами: через відкриття НД ***Query1*** або шляхом встановлення властивості ***Connected*** компонента ***Database1*** в значення ***True***.

Псевдонім для роботи з віддаленою БД зручно створювати за допомогою програми ***BDE Administrator***. Для сервера ***InterBase*** псевдонім має тип ***INTERBASE.*** Для псевдоніма потрібно завдати наступні параметри:

- Ім'я користувача ***USER NAME (SYSDBA).***
- Розміщення БД ***SERVER NAME*** (для локального сервера ***InterBase*** це шлях до БД, для віддаленого сервера цей шлях

прописаний в файлі ***hosts***, який розміщений в каталозі **WINDOWS**).

- Мовний драйвер ***LANGDRIVER*** (***Pdox ANSI Cyrillic***).

Для створення додатку в **Delphi** використаємо наступні компоненти:

DBGrid (з панелі інструментів ***Data Controls***), ***DataSourse*** (***Data Access***), ***IBTable, IBDatabase, IBTransaction*** (три останні з панелі інструментів ***InterBase***). Для компоненти ***DBGrid*** властивості ***DataSourse*** присвоїмо значення ***DataSourse1***, для компоненти ***DataSourse*** властивості ***DataSet*** присвоїмо значення ***IBTable1***, для компоненти ***IBTransaction*** властивості ***DefaultDatabase*** присвоїмо значення ***IBDataBase***, для компоненти ***IBDataBase*** властивості ***DefaultDatabase*** присвоїмо значення ***IBDataBase***, властивості ***DatabaseName*** присвоїмо шлях до БД (властивості ***LoginPromt*** присвоїмо значення ***False***), для компоненти ***IBTable*** властивості ***Database*** присвоїмо значення ***IBDatabase1***, властивості ***TableName*** присвоїмо ім'я таблиці, властивості ***Active*** присвоїти значення ***True***. Між властивостями компонентів ***IBTable*** та ***Table*** немає принципової різниці.

Зауваження. При використанні віддаленої бази шлях прописується у формі

'IP адрес серверу': 'Повний шлях до бази'.

Для підтримки цілісності БД використовується механізм транзакцій. Транзакція – це послідовність операцій, які виконуються для декількох НД. Вона переводить БД з одного цілісного стану в другий. Транзакція може бути *явною* та *неявною*. *Неявна* запускається автоматично при модифікації НД. Вона підтверджується методом ***Post*** для закріплення змін в НД, і відміняється методом ***Cancel***. *Явна* транзакція означає, що програміст має самостійно організувати операції зміни НД. Для реалізації механізму явних транзакцій **Delphi** надає спеціальні методи компонента ***DataBase***:

- ***StartTransaction***, який починає транзакцію. Після нього розташовуються інструкції, які складають транзакцію.
- ***Commit,*** який підтверджує транзакцію і всі зміни вступають в силу.
- При появі виключень викликається метод ***Rollback,*** який відміняє транзакцію і дії всіх операцій в рамках цієї транзакції.

Для контролю обміну даних між клієнтом і сервером служить компонент ***IBTransaction.***

Контрольні завдання:

Після засвоєння матеріалу лабораторної роботи слід вміти:

1. Створити базу даних ***InterBase***.
2. Організувати роботу БД в архітектурі "клієнт-сервер".
3. Організувати найпростіший контроль над транзакціями.
4. Створити додаток БД з даними аналогічними попереднім лабораторним роботам в архітектурі "клієнт-сервер". Організувати навігацію, пошук, сортування та фільтрацію даних.
5. Оформити короткий звіт про виконану роботу, внести в нього основні теоретичні відомості.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. **Пасічник В.В., Резніченко В. А.** Організація баз даних та знань. – К. Видавнича група BHV, 2006. – 384 с.

2. **Гофман В., Хомоненко А.** Delphi7 – СПб.: БХВ-Петербург, 2008. – 1216 с.

3. **Фаронов В**. Delphi 2005. Розработка приложений для баз данных и Интернета. – Питер, 2006. – 826 с.

4. **Архангельский А. Я.** Приемы програмирования в Delphi 2005. – М.: ООО «Бином-Пресс», 2004. – 848 с.

5. **Фаронов В**. Delphi 20006:учебный курс. – СПб: Питер, 2002. – 512 с.

6. **Бобровский С. И.** Delphi 7:учебный курс. – СПб,: Питер, 2003. – 736 с.

7. **Кандзюба С. П., Громов В.И.** Delphi 6/7. Базы данных и приложения. Лекции и упражнения. – СПб: ООО «ДиаСофтЮП», 2002. – 576 с.

ЗМІСТ