Національний університет “Києво-Могилянська Академія”

# Курсова робота

на тему:

“ **Visual C++. Бази даних.** ”

### Студентки 4-го курсу

департаменту

комп’ютерних технологій

## Фіялка Світлани

Науковий керівник:

Спасітєлєва С.О.

## Київ-1999

Зміст:

## Вступ

Програмування баз даних за допомогою ODBC

 Архітектура ODBC

 Концепції API ODBC

 Блок-схема типової ODBC-програми

 Програма установки ODBC

 Приклад програми ODBC

 Стандарт SQL та ODBC

 ODBC в MFS-застосуваннях

 Створення структури застосування за допомогою AppWizard

Об’єкти доступу до даних (DAO

 Класи DAO

 Побудова застосування DAO

Візуальні інструменти баз даних

Висновки

**Вступ .**

В першу чергу **Visual C++** - це компілятор C++, але це також і середовище, компоненти якого, взаємодіючи один з одним, спрощують процес розробки застосувань. Середовище Visual C++ пропонує великі можливості для програмування Windows-застосувань. Найхарактернішою його компонентою є бібліотека основних класів Microsoft (Microsoft Foundation Classes – **MFS**). Великий набір класів С++ інкапсулює основну частину API (Application Standart Interface) Win32 і пропонує могутню основу для написання типових застосувань. До складу Visual C++ входить **Microsoft Developer Studio Integrated Development Environment** – інтегроване середовище для розробки застосувань (IDE). **Visual Studio 97** – остання версія Microsoft Developer Studio – є ядром системи розробки Visual C++. Вона пропонує багато різних можливостей, надає доступ до багатьох компонент системи розроблювача Visual C++, а також взаємодіє з такими засобами розробки Micrisoft, як Visual J++ або Microsoft Network Library. Visual Studio підтримує інтерфейс для компонентів проектів, початкових файлів і файлів ресурсів, а також ряд інструментів для побудови, запуску і тестування застосувань, хоча компілятор C/C++ та інші компоненти середовища розробника можна використовувати і з командного рядка. Visual Studio пропонує інтегрований інтерфейс для таких основних інструментів Visual C++ як майстри. **Майстри (Wizards)**, такі як AppWizard, - це інструменти генерації структур застосувань. За допомогою таких майстрів можна створювати структури застосувань серверів або контейнерів компонентів OLE, застосувань ODBC та DAO, серверів автоматизації, елементів управління ActiveX.

Visual C++ пропонує широкі можливості для програмування баз даних, які, зокрема, полягають в розширеному керуванні джерел даних і відладці. В застосуваннях можна використовувати декілька технологій баз даних: **ODBC**(Open Database Connectivity) - Зв’язок відкритих баз даних, незалежний від постачальника механізм доступу до даних з різних джерел, **DAO**(Data Access Objects) – Об’єкти доступу до даних, застосовується для доступу к базам даних в Microsoft Visual Basic, Microsoft Access, Visual Basic for Applications, **ADO** (ActiveX Data Objects) – компактна технологія доступу до даних, орієнтована на Web, **OLE DB** – база даних OLE, набір специфікацій інтерфейсів компонентної об’єктної моделі COM.

Метою даної курсової роботи є розгляд основних можливостей програмування баз даних в Visual C++, зокрема використовуючи ODBC та DAO.

**Програмування баз даних за допомогою ODBC.**

**ODBC** або Open Database Connectivity (Зв’язок відкритих баз даних) – незалежний від постачальника механізм доступу до даних з різних джерел. Він є реалізацією специфікацій X/Open та SAG CLI (SQL Access Group Call Level Interface) – інтерфейс рівня виклику групи доступу SQL, яку виконала фірма Microsoft. ODBC пропонує функції для взаємодії з базами даних за допомогою мови програмування, наприклад, додавання, модифікування і видалення даних, отримання службової інформації про базу даних, таблиці та індекси.

*Архітектура ODBC.*

Архітектура ODBC складається з п’яти основних рівнів: **прикладний рівень, інтерфейс ODBC, диспетчер драйверів, драйвер та джерело даних***.*

 **Прикладний рівень** реалізує GUI (Graphical User Interface – Графічний інтерфейс користувача) та бізнес-логіку. Він написаний на мові програмування, такій як Java, Visual Basic або C++. Прикладна програма використовує функції з інтерфейсу ODBC для взаємодії з базами даних.

**Диспетчер драйверів** є частиною ODBC Microsoft. Він керує різними драйверами, що знаходяться в системі, виконуючи завантаження, спрямування викликів на потрібний драйвер і надання прикладній програмі інформації про драйвер, коли це необхідно. Оскільки одна прикладна програма може бути зв’язана з декількома базами даних, то диспетчер драйверів гарантує, що відповідна система керування базою даних отримує всі запити, що надходять до неї, та що всі дані з джерела даних будуть передані прикладній програмі.

**Драйвер** – та частина архітектури, яка все знає про будь-яку базу даних. Система драйверів ODBC працює не лише на платформі WINDOWS, але і на інших, зокрема UNIX, що перетворює API ODBC на могутній інструмент для створення платформно-незалежних застосувань. Крім того ODBC приводить використовуємий в застосуванні синтаксис SQL до єдиного стандарту, що дозволяє розробляти застосування незалежні від вида бази даних. Звичайно драйвер пов’язаний з конкретною базою даних, наприклад, драйвери Access, Oracle та драйвер SQL Server. Інтерфейс ODBC має набір функцій, таких як оператори SQL, управління з’єднанням, інформація про базу даних тощо. В обов’язки драйвера входить їх реалізація. Це означає, що в деяких базах даних драйвер повинен емулювати функції інтерфейсу ODBC, які не підтримуються системою керування базою даних. Він виконує роботу по розсиланню запитів в бази даних, отриманню відповідей і відправленню їх прикладній програмі. Для баз даних, що працюють в локальних мережах або Internet, драйвер підтримує мережний зв’язок.

**Джерело даних** в контексті ODBC може бути системою керування базою даних або просто набором файлів на жорсткому диску. Він може бути як простою базою даних Microsoft Access для невеликої фірми, так і багатосерверним зберігачем інформації про клієнтів телефонної компанії і їх розмовах.

*Коцепції API ODBC.*

Основною частиною системи ODBC є **драйвер**. Драйвери можуть бути однорівневі та багаторівневі. Однорівневі драйвери обробляють звертання ODBC і оператори SQL, а багаторівневі – обробляють звертання ODBC та передають оператори SQL джерелу даних (можливо, серверу, розташованому де-небудь в мережі). ODBC не вимагає, щоб драйвери підтримували всі функції цього інтерфейсу. Замість цього для драйверів визначаються рівні відповідності API та граматики SQL. Єдина вимога: якщо драйвер задовільняє деякому рівню, то він повинен підтримувати всі функції ODBC, визначені на цьому рівні, незалежно від того, чи підтримує їх база даних. Так як в обов’язки драйвера входить емуляція функцій ODBC, що не підтримуються системою керування базою даних, то інтерфейс ODBC не залежить від реалізації бази даних.

*Рівні відповідності API та граматики SQL для ODBC.*

|  |  |  |
| --- | --- | --- |
| Тип | Рівень відповідності | Опис |
| РівнівідповідностіAPI | Ядро | Всі функції із специфікації SAG CLI. Виділення і звільнення описувачів зв’язку, SQL-оператора і оточення. Підготовка і виконання операторів SQL. Отримання результату і службової інформації про результат. Отримання інформації про помилки. Здатність виконувати транзанкції. |
|  | Рівень 1 | Розширений набір 1 включає API ядра плюс можливості посилати і отримувати часткові набори даних, шукати інформацію в каталозі,отримувати інформацію про можливості драйвера і бази даних, тощо. |
|  | Рівень 2 | Розширений набір 2 включає рівень 1 плюс можливості обробляти масиви як параметри, можливість прокрутки курсору, виклик DLL транзанкцій, тощо. |
| РівнівідповідностіграматикиSQL | Мінімальнаграматика | Функції створення і видалення таблиць в мові визначення даних. Прості функції вибору, вставки, модифікації і видалення в мові маніпулювання даними. Прості вирази. |
|  | Граматика ядра | Відповідність специфікаціям SAG CAE 1992 на мінімальну граматику плюс зміна таблиць, створення і видалення індексу, створення і видалення логічних таблиць бази даних для DLL. Повний оператор SELECT для DDL. Повний Оператор SELECT для DML. Функціїї в виразах, наприклад, SUM and MAX. |
|  | Розширена граматика | Додаткові можливості, такі як зовнішні з’єднання, позиційовані модифікація і видалення, більше виразів і типів даних, виклики процедур. |

ODBC не є еквівалентом Embedded SQL (Вбудований SQL), який використовує оператори SQL в вихідних програмах, написаних на інших мовах. Схожі гібрідні програми перед передечею компілятору основної мови програмування оброблюються прекомпілятором. ODBC інтерпретує оператори SQL під час виконання. Основна програма не потребує прекомпіляції для виконання різних операторів SQL, як і не треба компілювати окремі версії програми для різних джерел даних.

*Блок-схема типової ODBC-програми.*

Перш ніж реалізувати оператори SQL, застосування ODBC повинно виконати ряд дій, щоб під’єднатись до джерела даних. Типові дії звертання ODBC:

 **SQLAllocEnv();** Виділяє середовище ODBC

 **SQLAllocConnect();** Виділяє пам’ять для підключення

 **SQLConnect();** Завантажує драйвер, підключає до джерела

 **SQLAllocStmt();** Виділяє пам’ять для оператора SQL

//Виконати оператори ODBC

SQLPrepare();

SQLExecute();

SQLBindCol();

SQLFetch();

…..

 **SQLFreeStmt();** Звільнює пам’ять, виділену для оператора

 **SQLDisconnect();** Перерозподіляє драйвер, відключає від джерела

 **SQLFreeConnect();** Звільнює пам’ять, виділену для підключення

 **SQLFreeEnv();** Звільнює середовище, перериває сеанс

Виклик SQLAllocEnv() ініціалізує бібліотеку ODBC і повертає дескриптор типу SQLHENV. Дескриптор типу SQLHDBC, що повертається функцією SQLAllocConnect(), використовується в наступних звертаннях до функцій ODBC для посилання на певне підключення. Одне застосування може підтримувати декілька відкритих підключень. Функція SQLConnect() шляхом завантаження драйверу і підключення до джерела даних встановлює з’єднання. Це звертання до функції має варіанти; наприклад, виклик SQLDriverConnect(), можна використовувати для підключення до джерел даних, які не встановлюються за допомогою програми початкової установки ODBC. SQLBrowseConnect() дозволяє застосуванню циклічно продивлятися джерела даних. Виділяючи пам’ять для оператора SQL за допомогою функції SQLAllocStmt() на окремому кроці, ODBC забезпечує механізм, при якому оператори можуть конструюватися і використовуватися один раз і кілька разів перш, ніж виділена пам’ять. Після цих чотирьох звертань, як правило, застосування ODBC створює звертання до бази даних для виконання операторів SQL. Воно може використовувати функцію SQLPrepare() для підготовки (компіляції) оператора SQL для виконання і функцію SQLExecute() для дійсного його виконання. В якості альтернативи виклику SQLPrepare() та SQLExecute() застосування можуть використовувати функцію SQLExecDirect() для виконання оператора SQL в одній дії. Спочатку стовпчики зв’язуються з змінними програми за допомогою SQLBindCol, потім ці змінні зчитуються після виконання SQLFetch() над рядком таблиці. Якщо даних більше нема, SQLFetch() повертає SQL\_NO\_DATA\_FOUND. В якості альтернативи використанню SQLBindCol() для пов’заних стовпців застосування можна використовувати функцію SQLGETDATA() для отримання даних з незв’язаних стовпців.

Коли його робота закінчена, застосування повинно звільнити задіяні ним ресурси ODBC.

В ODBC 3.0 функція SQLFreeHandle() використовується замість функцій ODBC 2.x SQLFreeEnv(), SQLFreeConnect(), SQLFreeHandle().

*Програма установки ODBC.*

**Програма установки ODBC** викликається за допомогою Control Panel (Панель керування) в Windows і використовується для регістрації джерел даних. Що саме є джерелом даних? Це залежить від драйвера. У випадку такого драйвера, як SQL Server, джерелом даних може бути база даних на сервері. В випадку драйвера Microsoft Access або Microsoft Excel база даних є файлом (MDB або XLS). В випадку Microsoft Text Driver база даних – це каталог, що містить текстові файли, які стосовно драйвера є таблицями в базі даних. Останні версії ODBC розрізняють три типи джерел даних. Джерело даних користувача – це джерело даних, яке бачить лише той, хто його створив. Джерело системних даних доступне всім користувачам даного комп’ютера. Джерело файлових даних - це джерело даних, специфікації якого зберігаються в файлі і можуть бути одночасно використані користувачами на різних комп’ютерах.

Для додавання джерела даних треба викликати програму установки ODBC в Control Panel і, вибравши необхідний тип джерела даних, вибрати драйвер і додати його кнопкою Add. В з’явившомуся діалоговому вікні “Create New Data Source” можна вибрати базу даних і встановити необхідні властивості драйвера. Основне діалогове вікно містить список всіх інстальованних джерел даних:

 джерело даних джерело системних джерело файлових

 користувача даних даних

 Діалогове вікно “ODBC Data Source Administrator”

*Приклад програми ODBC.*

Нище наведено текст простого застосування ODBC, яке зчитує рядки, що містяться в таблиці Excel. Коли доступ до таблиці Excel здійснюється з використанням драйвера Microsoft Excel ODBC, електронні таблиці відіграють роль таблиць бази даних, а рядки в таблиці – роль записів у ній.

Таблиця Excel складається з 4 стовпчиків, що містять інформацію про прізвище та ім’я студента, факультет, де він навчається, та рейтинг за осінній триместр 1998/1999 навч. рік.

Замість інсталяції цієї таблиці в якості джерела даних, використовуючи програму початкової установки ODBC, можна скористуватися можливостями функції SQLDriverConnect.

Застосування може бути скомпільовано з командного рядка введеннням:

**cl student.c odbc32.lib**. Файл student.xls (таблиця Excel) повинен бути доступним з поточного каталогу.

#include <windows.h>

#include <sql.h>

#include <sqlext.h>

#include <string.h>

#define CONNSTR «DBQ=Student.XLS;DRIVER={Microsoft Excel Driver (\*.xls)}»

#define CONNLEN (sizeof(CONNSTR)-1)

#define SQLTRY(x,y) \

 { \

 rc = y; \

 if (rc != SQL\_SUCCESS) \

 { \

 char szState[6]; \

 char szMsg[255]; \

 SDWORD sdwNative; \

 SWORD swMsgLen; \

 SQLError(hEnv, hDBC, hStmt, szState, &sdwNative, szMsg, sizeof(szMsg), &swMsgLen); \

 printf(«Error %d performing %s\nSQLState = %s\nSQL message = %s\n», rc, x, szState, szMsg); \

 goto Terminate; \

 } \

 }

void main(void)

{

 SQLHENV hEnv = 0;

 SQLHDBC hDBC = 0;

 SQLHSTMT hStmt = 0;

 SQLCHAR szConnStr[255];

 SQLCHAR szStmt[255];

 SQLCHAR szName[255];

 SQLCHAR szFaculty[255];

 long nRating;

 SWORD cbConnStr;

 RETCODE rc;

 SDWORD sdwNLen;

 SDWORD sdwFLen;

 SDWORD sdwRLen;

 int i;

 char szResult[1000];

 SQLTRY(«SQLAllocEnv», SQLAllocEnv(&hEnv))

 SQLTRY(«SQLAllocConnect», SQLAllocConnect(hEnv, &hDBC))

 SQLTRY(«SQLDriverConnect», SQLDriverConnect(hDBC, NULL, CONNSTR, CONNLEN, szConnStr, sizeof(szConnStr), &cbConnStr, SQL\_DRIVER\_NOPROMPT))

 SQLTRY(«SQLAllocStmt», SQLAllocStmt(hDBC, &hStmt))

 **sprintf(szStmt, «SELECT \* FROM [Sheet1$]»);**

 SQLTRY(«SQLPrepare», SQLPrepare(hStmt, szStmt, strlen(szStmt)))

 SQLTRY(«SQLBindCol», SQLBindCol(hStmt, 1, SQL\_C\_CHAR, (PTR)szName, sizeof(szName), &sdwNLen))

 SQLTRY(«SQLBindCol», SQLBindCol(hStmt, 2, SQL\_C\_CHAR, (PTR)szFaculty, sizeof(szFaculty), &sdwFLen))

 SQLTRY(«SQLBindCol», SQLBindCol(hStmt, 3, SQL\_C\_SLONG, (PTR)&nRating, sizeof(nRating), &sdwRLen))

 SQLTRY(«SQLExecute», SQLExecute(hStmt))

 for (i = 1; (rc = SQLFetch(hStmt)) == SQL\_SUCCESS; i++)

 {

 printf(«Record #%d\tName: %s\tFaculty: %s\tRating: %d\n», i, szName, szFaculty, nRating);

 }

 if (rc != SQL\_NO\_DATA\_FOUND)

 {

 SQLTRY(«SQLFetch», rc)

 }

 printf(«Successfully completed.\n»);

Terminate0:

 if (hStmt) SQLFreeStmt(hStmt, SQL\_CLOSE);

 if (hDBC) SQLDisconnect(hDBC);

 if (hDBC) SQLFreeConnect(hDBC);

 if (hEnv) SQLFreeEnv(hEnv);

 SQLTRY(«SQLAllocEnv», SQLAllocEnv(&hEnv))

 SQLTRY(«SQLAllocConnect», SQLAllocConnect(hEnv, &hDBC))

 SQLTRY(«SQLDriverConnect», SQLDriverConnect(hDBC, NULL, CONNSTR, CONNLEN, szConnStr, sizeof(szConnStr), &cbConnStr, SQL\_DRIVER\_NOPROMPT))

 SQLTRY(«SQLAllocStmt», SQLAllocStmt(hDBC, &hStmt))

 **sprintf(szStmt, «SELECT \* FROM [Sheet1$] WHERE Rating>91 ORDER BY Rating DESC»);**

 SQLTRY(«SQLPrepare», SQLPrepare(hStmt, szStmt, strlen(szStmt)))

 SQLTRY(«SQLBindCol», SQLBindCol(hStmt, 1, SQL\_C\_CHAR, (PTR)szName, sizeof(szName), &sdwNLen))

 SQLTRY(«SQLBindCol», SQLBindCol(hStmt, 2, SQL\_C\_CHAR, (PTR)szFaculty, sizeof(szFaculty), &sdwFLen))

 SQLTRY(«SQLBindCol», SQLBindCol(hStmt, 3, SQL\_C\_SLONG, (PTR)&nRating, sizeof(nRating), &sdwRLen))

 SQLTRY(«SQLExecute», SQLExecute(hStmt))

 for (i = 1; (rc = SQLFetch(hStmt)) == SQL\_SUCCESS; i++)

 {

 printf(«Record #%d\tName: %s\tFaculty: %s\tRating: %d\n», i, szName, szFaculty, nRating);

 }

 if (rc != SQL\_NO\_DATA\_FOUND)

 {

 SQLTRY(«SQLFetch», rc)

 }

 printf(«Successfully completed.\n»);

Terminate:

 if (hStmt) SQLFreeStmt(hStmt, SQL\_CLOSE);

 if (hDBC) SQLDisconnect(hDBC);

 if (hDBC) SQLFreeConnect(hDBC);

 if (hEnv) SQLFreeEnv(hEnv);

}

Простий макрос SQLTRY використовується для повідомлення про помилки.

Після обов’язкових звертань до функцій SQLAllocEnv та SQLAllocConnect програма викликає SQLDriveConnect. Цей виклик робить можливим відкриття таблиці, яка не встановлювалась з використанням програми початкової установки ODBC і робить це без відображення інтерфейса користувача. Для відкриття таблиці використовуються константи CONNSTR та CONNLEN. Як тільки підключення до бази даних успішно завершено, виконуються послідовно два оператори SQL:

1. **SELECT \* FROM [Sheet$]** - Вибрати всі записи з таблиці Student.
2. **SELECT \* FROM** **[Sheet$] WHERE Rating>91 ORDER BY Rating DESC –** Вибрати записи з таблиці Student про тих студентів, що мають рейтинг більший за 91, та впорядкувати виведений список за зменшенням рейтингів студентів.

Ім’я **Sheet$ -** це ім’я, яке надається драйвером для першої таблиці в робочій книзі Excel. Оператор SQL використовується для отримання полів всіх записів.

Наступні 4 звертання прив’язують змінні до стовпчиків таблиці. Таке призначення функції SQLBindCol. Після послідовного отримання записів, значення полів переміщуються в ці змінні.

Самі записи отримуються за допомогою функції SQLFetch і відображаються з використанням printf. Функція SQLFetch викликається до тих пір, поки значення, що повертається нею, не відрізняється чим-небудь від SQL\_SUCCESS. Значення, що повертається, SQL\_NO\_DATA\_FOUND показує, що отриманий останній запис, все інше є помилкою і обробляється відповідним чином.

Програма завершується обов’язковими звертаннями до функцій SQLFreeStmt, SQLDisconnect, SQLFreeConnect та SQLFreeEnv для звільнення ресурсів і закінчення зв’язку з джерелом даних.

При запуску цієї програми забезпечується такий вивід:



*Стандарт SQL та ODBC.*

Синтаксис для ODBC заснований на стандарті ANSI SQL-92.

Майже кожна СУБД використовує свій власний діалект SQL, синтаксис якого може трохи відрізнятися. Кожен діалект може підтримувати різний набір властивостей. Драйвер ODBC для конкретної бази даних забезпечить необхідний переклад загального SQL на специфічний діалект, що використовується в певній СУБД.

Існує три основні групи операторів SQL:

* мова визначеня даних DDL (Data Definition Language) використовується для задання структури бази даних і створення її об’єктів;
* мова керування даними DCL (Data Control Language) – для роботи з правами користувача на конкретні об’єкти;
* мова маніпулювання даними DML (Data Manipulation Language) – для виконання інших операцій таких, як додавання і модифікація даних, а також виконання запитів.

Всі ці мови є частиною SQL, а не є окремими мовами, хоча в більшості застосувань використовуються лише оператори, що відносяться до конкретної групи.

В **DDL** для роботи з об’єктами використовуються такі оператори SQL:

* *CREATE* – створює новий об’єкт;
* *ALTER* – модифікує існуючий об’єкт;
* *DROP-* видаляє об’єкт.

Ці оператори використовуються для роботи з різними об’єктами бази даних таких, як таблиці, індекси і представлення таблиць. Для кожного з цих типів об’єктів використовується різний синтаксис оператора.

В таблицях – основних об’єктах реляційних базах даних – можуть міститися дані різних типів. Типи даних SQL, згруповані згідно рівням підлеглості ODBC SQL.

Мінімальний рівень:

* CHAR (n) – символьний рядок з n символів фіксованої довжини.
* VARCHAR (n) – символьний рядок змінної довжини, але не більше n символів.
* LONG VARCHAR – символьний рядок змінної необмеженої довжини.

Базовий рівень:

* DECIMAL (p,s) або NUMERIC (p,s) – значення з плавучою комою.
* SMALLINT- двохбайтове ціле.
* INTEGER – чотрьохбайтове ціле.
* REAL – чотирьохбайтове значення з плавучою комою.
* FLOAT або DOUBLE PRECISION – восьмибайтове значення з плавучою комою.

Розширений рівень:

* BIT – один біт.
* TINYINT – однобайтове ціле.
* BIGINT – восьмибайтове ціле.
* BINARY (n) – двійкове поле фіксованої довжини, що складається з n байт.
* VARBINARY (n) – двійкове поле змінної довжини, що не перевищує n байт.
* LONG VARBINARY- двійкове поле змінної необмеженої довжини.
* DATE- значення дати.
* TIME – значення часу.
* TIMESTAMP – значення, що містить час і дату.

**Створення таблиці.**

*CREATE TABLE Student (*

 *StudID UNTEGER,*

 *StudName VARCHAR (20) )*

Стовпчики таблиці можна довизначити, якщо додати після типу даних додаткові модифікатори. Найчастіше використовується модифікатор NOT NULL, який вказує на те, що значення для цього стовпця повинно бути задано. Можна також вимагати, щоб конкретне поле в таблиці не повторювалось. Для цього використовується модифікатор UNIQUE. Можна також задати для стовпця значення, що буде використовуватися по замовченню, якщо не було задано явних значень. Для цього служить модифікатор DEFAULT.

**Видалення таблиці.**

*DROP TABLE Student*

В опрераторі видалення можуть використовуватися ключові слова CASCADE та RESTRICT. Якщо в операторі вказано ключове слово CASCADE, то при видаленні таблиці видаляються також всі представлення або обмежувачі цілостності, пов’язані з цією таблицею. Модифікатор RESTRICT не дозволить видалити таблицю, якщо на неї є посилання в яких-небудь представленнях або обмежувачах цілостності.

**Зміна таблиці.**

Додавання стовпців:

*ALTER TABLE Student ADD COLUMN Course INTEGER*

Видалення стовпців:

*ALTER TABLE Student DROP COLUMN Course*

Як і при видаленні таблиці, тут можна використовувати ключові слова CASCADE та RESTRICT. При включенні модифікатору CASCADE видаляються також всі представлення і обмежувачі, які посилаються на видаляємий стовпець. Використання модифікатора RESTRICT запобігає видаленню стовпця, якщо на нього є посилання в представленнях або обмежувачах цілостності посилань.

**Індекси.**

Це об’єкти баз даних, що забезпечують більш ефективний доступ до рядків таблиці. Індекс продивляється рядки таблиці на основі значень певного стовпця або декількох стовпців. Індекси можуть значно підвищити продуктивність бази даних при виконанні певних типів операцій пошуку, проте для їх підтримання треба великий обсяг динамічної пам’яті. Якщо для одної таблиці створити багато різних індексів, продуктивність всього застосування значно зменшиться.

*CREATE [UNIQUE] INDEX StudIndex ON Student (StudID) [ASC, DESC]*

ASC – в зростаючому порядку

DESC – в спадаючому порядку

Змінити індекс неможливо, а можна лише видалити його.

*DROP INDEX StudIndex*

**Представлення.**

Педставлення таблиць, або просто представлення можна розглядати як вид віртуальної таблиці. З точки зору застосувань користувачів, представлення таблиці виглядає як і сама таблиця. Проте в представленні зберігаються не його власні дані, а лише посилання на дані, що містяться в таблицях. В цілому представлення використовуються для відображення даних, що містяться в таблицях, різними способами зручними для користувача. Представлення можна використовувати для відображення підмножини стовпців однієї або декількох таблиць, що віповідають деякому типу запита. Це дозволяє представляти таблицю з потрібними даними, хоча в дійсності стуктура бази даних може бути іншою, сформованою виходячи з міркувань цілостності та продуктивності. Представлення таблиць зручні для забезпечення безпеки. За допомогою представлення можна забезпечити доступ користувачів до деякої підмножини стовпців в таблиці, залишаючи захищеними інші стовпці. Представлення служать і для забезпечення логічної незалежності даних, завдяки чому зміни занесені в реальні таблиці баз даних не відображаються на застосуваннях користувача. Якщо таблиця в базі даних буде змінена, то можна буде просто настроїти представлення у відповідності з новою структурою таблиці і не треба буде повертатися і вносити зміни в кожне написане застосування.

*CREATE VIEW MyView AS SELECT \* FROM TABLE MyTable*

Представлення відображає всі стовпці таблиці

*CREATE VIEW MyView (Id, Name) AS SELECT (StudID, Studname) FROM Student*

Існуюче представленя не можна модифікувати, але можна його видалити.

*DROP VIEW MyView*

**Мова керування даними DCL.**

Дозволяє працювати з правами користувачів на об’єкти в базі даних.

Мова SQL дозволяє присвоїти групі користувачів певні права на деякий об’єкт. Права можуть бути такими:

* SELECT дозволяє запитувати дані
* INSERT дозволяє додавати нові рядки
* DELETE дозволяє видаляти рядки
* UPDATE дозволяє змінювати існуюючі рядки
* REFERENCES це право потрібне лише в тому випадку, коли користувач намагається модифікувати таблицю, що має обмеження цілостності посилань, які зсилаються на стовпці іншої таблиці. Такий користувач повинен мати право REFERENCES на стовпці, що використовуються з обмеженнями.

Права присвоюються:

*GRANT SELECT ON Student TO PUBLIC*

*GRANT SELECT, INSERT ON Student TO Sveta, Lena*

Права UPDATE та REFERENCES можуть поширюватися на окремі стовпці.

*GRANT UPDATE (Raiting, Credit) ON Student TO Sveta*

Відмінити права на об’єкти можна за допомогою оператора REVOKE:

*REVOKE SELECT ON Student FROM Lena*

В операторі REVOKE можна використовувати модифікатори CASCADE та RESTRICT. Якщо відміняються права користувача на деяку таблицю, то доцільно відмінити його права на всі представлення, що потребують доступу до всієї таблиці. Зробити це можна за допомогою CASCADE. Модифікатор RESTRICT не дозволяє відмінити право доступу, яке вимагається згідно іншим правам, що присвоєні цьому користувачу.

**Мова маніпулювання даними DML.**

Основні типи операторів:

* SELECT
* INSERT
* DELETE
* UPDATE

*SELECT* – основа SQL. Використовується кожен раз при отриманні даних. Базовий синтаксис оператора SELECT:

SELECT \* FROM Student

Інші:

SELECT \* FROM Student WHERE Raiting>91 AND Credit<120

SELECT \* FROM Student WHERE NOT Course=4

SELECT \* FROM Student WHERE (Raiting/Credit)<1

В SQL є спеціальний оператор для порівняння символьних рядків. Предикат *LIKE* вибирає рядки на основі визначеного шаблону. В шаблон можуть входити звичайні символи, а також символи % та \_. Рядок символів довільної довжини можна поставити у відповідність шаблону %, а один символ - \_.

Вибір всіх рядків, що містять рядок Database в полі Title

*SELECT ChapterNum FROM Chapters WHERE Title LIKE ‘%Database%’*

Пошук заголовків, в яких починаючи з другої позиції міститься рядок ata

*SELECT ChapterNum FROm Chapters WHERE Title LIKE ‘\_ata%’*

Для спрощення оператору WHERE використовується предикати *IN* та *BETWEEN*:

*SELECT StudID FROM Student WHERE Course IN (4, 3)*

*SELECT StudName FROM Student WHERE Raiting BETWEEN 91 AND 100*

В багатьох задачах треба знайти всі можливі значення певного поля на деякій множині рядків. Зробити це можна за допомогою ключового слова *DISTINCT*, яке запобігає дублюванню елементів в вихідній множині результатів.

*SELECT DISTINCT Course FROM Student*

По замовченню рядки, що повертаються після виконання запиту, подаються в довільному порядку. Змінити це можна за допомогою оператора *ORDER BY*, який дозволяє вказати один або декілька стовпців, що будуть використовуватися для сортування повертаємих рядків. Для кожного стовпця можна вказати модифікатор ASC або DESC.

*SELECT StudID FROM Student ORDER BY StudID ASC*

Якщо в операторі ORDER BY вказати декілька стовпців, то повертаємі рядки будуть відсортовані спочатку за першим стовпцем. Потім рядки, що містять однакові значення в першому полі, будуть відсортовані ще і за другим стовпцем.

Сортування рядків за зростанням значень в стовпці Salary, а рядки з однаковими значеннями в цьому полі – за спаданням ідентифікаційних номерів EmpID:

*SELECT EmpID, Salary FROM Employee ORDER BY Salary ASC, EmpID DESC*

Дуже часто треба вираховувати значення на основі даних з усіх рядків, що повертаються, наприклад загальну суму значень в деякому стовпці всіх повертаємих рядків. Такі типи розрахунків можна виконувати за допомогою таких функцій:

* AVG – повертає середнє значення в стовпці.
* COUNT – підраховує кількість повертаємих рядків.
* MAX – повертає максимальне значення для деякого стовпця в результуючій множині.
* MIN – повертає мінімальне значення для деякого стовпця в результуючій множині.
* SUM – підраховує загальну суму всіх значень для деякого стовпця в результуючій множині.

Задання значень дати і часу викликає особливий інтерес. Як правило, в різних системах керування базами даних використовуються різні, а тому, несумісні, способи представлення літералів для таких значень. ODBC забезпечує переносимий синтаксис для таких значень.

Для літералів дати використовуються один з наступних форматів:

{d ‘yyyy-mm-dd’}

Для літералів часу:

{t ‘hh:mm:ss’}

Для представлення значень дати і часу:

{ts ‘yyyy-mm-dd hh:mm:ss’}

**Функції SQL.**

В SQL, що використовується для ODBC, визначено багато функцій, які можна використовувати при побудові запитів або в параметрі списку вибору, або в операторі WHERE. Ці функціі можна розподілити на 5 груп: числові, строкові, функції дати та часу, функції системи та функції перетворення типів даних.

Синтаксис ODBC визначає таку форму запису при виклику функції в SQL:

*{fn ім’я функції}*

**Строкові функції**

Деякі функції для роботи з рядками в ODBC SQL:

* CONCAT(вираз1, вираз2) – приєднання *вираз1* до *вираз2*;
* INSERT(вираз1, початок, довжина, вираз2) – повертає *вираз1*, в якому видалена *довжина* символів, починаючи з символу *початок*, замість яких, починаючи з символу *початок*, вставлено *вираз2*;
* LCASE(вираз) – повертає *вираз*, символи якого перетворені на відповідні з нижнього регістру;
* LENGTH (вираз) – довжина *вираз* в символах;
* POSITION (вираз1, вираз2) – повертає позицію *вираз1* в *вираз2*;
* UCASE(вираз) – повертає *вираз*, всі символи якого перетворені на відповідні верхнього регістру.

**Числові функції.**

Деякі числові функції в ODBC SQL:

* ABS(вираз) – повертає абсолютне значення *вираз*;
* MOD(вираз1, вираз2) – повертає остачу від ділення *вираз1* на *вираз2*;
* POWER(вираз1, вираз2) – повертає *вираз1* в степені *вираз2*;
* RAND(вираз) – повертає випадкове число. *Вираз* задає кількість випадкових чисел;
* ROUND(вира1, вираз2) – повертає *вираз1* округлене з точністю до *вираз2* знаків після коми;
* SQRT(вираз) – повертає квадратний корінь з *вираз*.

**Функції дати та часу.**

Деякі функції дати та часу в ODBC SQL:

* CURDATE() – повертає поточну дату;
* CURTIME() – повертає поточний час;
* NOW() – повертає поточний час і дату в форматі дата/час;

**Системні функції.**

Деякі системні функції в ODBC SQL:

* DATABASE() – повертає ім’ бази даних, що використовується в поточному сеансі;
* USER() – повертає ім’я користувача бази даних, що використовується в поточному сеансі;

**Перетворення типів даних.**

Функція перетворення типів даних в ODBC SQL:

*CONVERT(вираз, тип\_даних)*

Параметр *тип\_даних* може приймати такі значення:

* SQL\_BINARY
* SQL\_CHAR
* SQL\_DECIMAL
* SQL\_DOUBLE
* SQL\_FLOAT
* SQL\_INTEGER
* SQL\_LONGVARCHAR
* SQL\_NUMERIC
* SQL\_REAL
* SQL\_SMALLINT
* SQL\_TYPE\_DATE
* SQL\_TYPE\_TIME
* SQL\_TYPE\_TIMESTAMP
* SQL\_VARCHAR

**Об’єднання.**

Операція об’єднання повертає рядки, що сформовані з даних двох або більше різних таблиць і об’єднані в процесі виконання запиту.

*SELECT Name, Prices FROM Products, Prices WHERE Products.ID=Prices.ID*

При виконанні цього оператора будуть виділені всі рядки з таблиці Products, що відповідають рядкам з таблиці Prices. Це так зване внутрішнє об’єднання. Для того, щоб створити запит, який повертає рядки, що не мають відповідності в одній з цих таблиць, треба використовувати зовнішнє об’днання.

Вибрати рядки для кожного працівника, зокрема тих, хто ще не відноситься ні до якого відділу.

*SELECT \* FROM*

*{oj Employee LEFT OUTER JOIN Department ON*

*Employee.DeptNum=Department.DeptNum}*

LEFT OUTER JOIN використовується для того, щоб гарантувати, що в результуючій множині будуть представлені всі рядки з лівої таблиці (Employee). Щоб включити всі рядки з правої таблиці можна використовувати RIGHT OUTER JOIN, а для вибору всіх рядків з обох таблиць – FULL OUTER JOIN.

**Вкладені запити.**

Вкладений запит може використовуватися з предикатом EXISTS (NOT EXISTS).

Вибрати список відділів, що не укомплектовані працівниками:

*SELECT DeptName FROM Department*

*WHERE DeptNum NOT EXISTS (SELECT \* FROM Employee)*

Повертається список відділів, номера яких не містяться в жодному запису таблиці Employee.

Можна також використовувати предикати IN та NOT IN.

Дуже часто вкладені запити використовуються для операцій порівняння з клячовими словами ANY та ALL.

Скласти список всіх студентів департаменту комп’ютерних технологій, у яких рейтинг вищий, ніж у студентів природничого факультету.

*SELECT Student\_DCSS.Name FROM Student\_DCSS*

*WHERE Student\_DCSS.Rating >ALL*

*(SELECT Student\_FNS.Rating FROM Student\_FNS)*

**Об’єднані запити**.

SQL дозволяє отримати один результат через об’єднання двох незалежних запитів шляхом їх комбінації за допомогою ключового слова UNION.

*SELECT \* FROM Students\_DCSS*

*UNION*

*SELECT \* FROM Students\_FNS*

**Оператор INSERT.**

За допомогою оператора INSERT можна вставити дані в певне місце в базі даних.

*INSERT INTO Students\_vipuskniki*

*VALUES (1, ‘Sveta Fiyalka’, ‘DCSS’, 95*)

**Оператор DELETE.**

*DELETE FROM Student WHERE Student.Rating<61*

**Оператор UPDATE.**

За допомогою цього оператора можна змінювати значення в існуючих рядках бази даних:

*UPDATE Employee SET Salary+100*

*ODBC в MFS-застосуваннях.*

Використання ODBC значно спрощується за допомогою **Microsoft Foundation Classes Library** (Бібліотека основних класів Microsoft). Прості застосування, що отримують доступ к таблицям через ODBC, можуть бути створені всього лише декількома натисненнями кнопки миші з використанням мастерів AppWizard та ClassWizard. Існує декілька класів MFS, які підтримують доступ до баз даних і наборам записів.

**Класи ODBC в MFS.**

Найголовніші класи, що надаються MFS для підтримки застосувань ODBC – це класи **CDatabase** та **CRecordset**. Клас CDatabase представляє з’єднання з джерелом даних. Його змінна m\_hdbc представляє дескриптор з’днання ODBC. Функції Open та Close можна використовувати для установки або закриття з’єднання з джерелом даних. Інші функції використовуються для установки або отримання параметрів з’єднання. До таких функцій належать: GetConnect(повертає рядок з’єднання ODBC), IsOpen, GetDatabaseName, CanUpdate, CanTransact, InWaitForDataSource, SetLoginTimeout та SetSynchronousMode. По замовченню для доступу до джерела даних клас CDatabase використовує асинхронний режим. Виконувана асинхронна дія може бути перервана викликом функції Cancel.

Обробка транзанкцій підтримується функціями BeginTrans та Rollback.

Клас CDatabase надає також дві перекриваємі функції. Функція OnSetOptions використовується для установки стандартних опцій з’єднання. Функція OnWaitForDataSource викликається системою для повідомлення про час обробки при виконанні довгої операції.

Функцію ExecuteSQL можна використовувати для безпосереднього виконання оператора SQL. Цей оператор не використовується разом з операторами SQL, що повертають записи даних.

Клас CRecordset інкапсулює функціональні можливості оператора ODBC SQL і набору рядків, що повертаються оператором. Змінні-члени цього класу ідентифікують дескриптор оператора ODBC, число полів і параметрів в наборі даних, об’єкт CDatabase, за допомогою якого набір записів підключен до джерела даних, і два рядки, що відповідають умовам WHERE та ORDER BY.

Двома основними типами наборів записів є динамічні і статичні набори. Тип набору записів вказується при виклику функції CRecordset:Open. Статичні набори представляють статичний варіант представлення даних. Це найбільш зручно для таких задач, як створення звіту. Динамічні набори представляють динамічний варіант представлення даних, що відображає зміни,виконані іншими користувачами або за допомогою інших наборів записів застосування.

Коли набір записів відкривається викликом функції Open, відбувається доступ до таблиці і запит, що представляється наьором записів, виконується. Набір записів і пов’язаний з ним дескриптор можуть бути закриті викликом функції Close.

Атрибути набору записів можна отримати через виклик функцій CanAppend, CanRestart, CanScroll, CanTransact, CanUpdate, GetRecordCount, GetTableName, GetSQL, IsOpen, IsEOF, IsBOF та IsDeleted.

Переміщення по набору записів можна отримати за допомогою функцій Move, MoveFirst, MoveLast, MoveNext та MovePrev.

Дії з набором записів можуть відбуватися викликом функцій Addnew, Delete, Edit або Update.

Об’єкт типу CRecordset ніколи не використовується безпосередньо. Необхідно отримати клас, похідний від CRecordset, і додати змінні, що відповідають оплям таблиці, яку представляють набір записів. Потім треба перекрити функцію DoFieldExchange набору записів; ця функція за допомогою функцій обміну полями записів RFX\_(Record Field Exchange) повинна сприяти обміну даних між змінними класу та полями в базі даних. Ці функції за синтаксисом аналогічні функціям обмуну даними діалогу (dialog data exchange – DDX\_).

*Функції RFX\_*

|  |  |  |
| --- | --- | --- |
| Ім’я функції | Тип поля | Тип SQL ODBC |
| FX\_Binary | CByteArray | SQL\_BINARY, SQL\_LONGVARBINARY,SQL\_VARBINARY |
| RFX\_Bool | BOOL | SQL\_BIT |
| RFX\_Byte | BYTE | SQL\_TINYINT |
| RFX\_Date | CTime | SQL\_DATE, SQL\_TIME, SQL\_TIMESTAMP |
| RFX\_Double | double | SQL\_DOUBLE |
| RFX\_Int | int | SQL\_SMALLINT |
| RFX\_Long | LONG | SQL\_INTEGER |
| RFX\_LongBinary | CLongBinary | SQL\_LONGVARCHAR |
| RFX\_Single | float | SQL\_REAL |
| RFX\_Text | Cstring | SQL\_CHAR, SQL\_DECIMAL, SQL\_LONGVARCHAR, SQL\_NUMERIC,SQL\_VARCHAR |

 Обмін полями підтримується за допомогою класу CfieldExchange. Об’єкт цього класу містить інформацію про поля, обмін якими повинен відбуватися при виклику функції DoFieldExchange набору записів.

Клас CRecordView – це клас перегляду, похідний від CformView, розробленого спеціально для відображення в формах записів баз даних. Об’єкти типу CRecordView використовують функції обміну даними діалогу (DDX) та обміна полями записів (RFX) для підтримки переміщення даних між формою і джерелом даних. Об’єкти, похідні від CRecordView використовуються разом з об’єктами, похідними від CRecordset.

Операції ODBC використовують клас CDBException для повідомлення про помилки шляхом механізму MFC-виключень.

*Створення структури застосування ODBC за допомогою AppWizard.*

Перш ніж MFS-застосування ODBC буде створено за допомогою AppWizard, необхідно ідентифікувати джерело даних, яким буде оперувати застосування. Джерело даних повинно бути ідентифіковано і інстальовано за допомогою програми установки ODBC.

Нехай джерелом даних буде база даних Microsoft Access. Для доступу до цього файлу потрібен драйвер Microsoft Access Driver.

База даних Student складається з 3 таблиць, що містять інформацію про студента, предмети та оцінки студента з цих предметів. В базі даних Access також створено запити, які оперують з даними бази.

Щоб створити структуру застосування ODBC треба запустити майстер AppWizard. Нехай проект буде мати назву - MFSDB.

Проекти, створені таким чином, можуть бути однодокументними, багатодокументними або діалоговими застосуваннями. Однодокументне застосування може представляти користувачу в будь-який момент лише один файл. Найкращим прикладом такого застосування є текстовий редактор Notepad в Windows. Багатодокументне застосування може одночасно представляти декілька документів, кожний у своєму вікні. Більшість текстових редакторів, таких як Microsoft Word, є багатодокументними застосуваннями. Інтерфейс користувача діалогового застосування являє собою одне діалогове вікно. Такі застосування використовуються, якщо взаємодію з користувачем можна провести в одному шаблоні діалогового вікна. Прикладом діалогового застосування є Таблиця символів (Character Map) Windows.

На першому кроці вказується, що проект буде однодокументним. На другому кроці вказуються опції бази даних, яку буде використовувати дане застосування. В якості джерела даних встановлюється Microsoft Access. Можна також вказати тип набору записів. Статичний набір записів (Snapshot) представляє статичний варіант проедставлення даних. Динамічний нібір записів (Dynaset) перехоплює лише індекс даних, таким чином, ці набори можуть відображати зміни, що виконані в записах після створення набору записів (але не додавання чи видалення яких-небудь записів). Набір записів у вигляді таблиці допускає більш конкретне маніпулювання таблицями в реляційній базі даних. В той час як статичні набори представляють більш безпечний вибір, динамічні набори забезпечують переваги в плані продуктивності.

При створенні проекту майстер AppWizaRD створює такі класи:



Новий клас CMFSDB є похідним від CRecordset. При об’явленні цього класу можна побачити, що майстер AppWizard не тільки створив клас, але й додав змінні-члени, що відображають поля вказаної таблиці бази даних. Ці змінні також відображаються в файлі реалізації класу, в конструкторі, а також – в функції DoFieldExchange. Остання викликається MFS для обміну даними між змінними набору записів і відповідними стовпцями в таблиці бази даних.

Такий створений проект представляє собою пусте діалогове вікно. Для його зміни треба додати елементи керування і скористатися майстером ClassWizard для додавання відповідних змінних-членів класів.

До діалогового вікна можна додати статичні елементи керування та елементи редагування, які будуть відображати необхідну інформацію, вибіка якої буде здійснюватися запитом:

**SELECT DISTINCT student.Name, subject.Subject**

**FROM (Mark INNER JOIN student ON Mark.Name = student.Name) INNER JOIN subject ON Mark.Subject = subject.Subject**

**WHERE (((Mark.Mark)<61));**

**Вибрати всіх студентів, що мають академзаборгованості.**

Даний запит можна реалізувати безпосередньо в середовищі Visual C++ за допомогою функції CRecordset::GetDefaultSQL. ЇЇ змінна – m\_strFilter відповідає умові SQL – WHERE.

**Cstring CMFSDBSet::GetDefaultSQL()**

**{**

 **m\_strFilter=\_T(“[Mark].[Mark]<61”);**

 **return \_T(“[Mark],[student]”);**

**}**

Кінцевий вигляд застосування:

За допомогою Component Galery до проекта було ще додано Tips of the Day:

**Об’єкти доступу до даних.**

**Data Access Objects** (Об’єкти доступу до даних) – один з останніх виробів Microsoft в технології доступу до баз даних. Ця технологія застосовується до баз даних в Microsoft Visual Basic, Microsoft Access та Visual Basic for Application; починаючи з Visual C++4, за допомогою набору спеціалізованих класів MFS ця технологія доступна і для програміста Visual C++.

DAO надає можливість доступу і маніпулювання базами даних за допомогою машини баз даних Microsoft Jet. За допомогою Jet-машини можна отримувати доступ до локальних і віддалених базам даних через драйвери ODBC.

Технологія DAO базується на OLE. Вона значно спрощується класами DAO в MFS.

Багато функцій DAO використовують оператори мови SQL. Можна скористатися SQL-оператором SELECT для отримання даних з бази даних або SQL-операторами UPDATE, INSERT та DELETE для зміни вмісту бази даних. Найпростіший спосіб створення SQL-операторів для використання з об’єктами DAO- створення запиту з Microsoft Access, збереження в базі даних та доступ до нього за допомогою об’єкта QueryDef.

Visual C++ запезпечує інтенсивну підтримку для побудови застосувань DAO за допомогою майстера AppWizard. В додаток до ODBC майстер AppWizard дозволяє створювати застосування, засновані на класах DAO.

*Класи DAO.*

Існує 5 основних та 2 допоміжних класи, пов’язаних з DAO.

 Всі об’єкти DAO похідні від DBEngine; більш того, всі об’єкти бази даних – похідні від об’єктів Workspace. Проте якщо не треба маніпулювати захищеними базами даних, як правило, нема необхідності звертатися до жодного з них, а достатньо первинний об’єкт Workspace визивати для всіх транзакцій.

Об’єкти бази даних і набору записів достатньо очевидно представляють бази даних і набори виборок (таблиці, набори записів або динамічні набори) в ціх базах даних.

Об’єкти визначення запитів (QueryDef) використовуються для виконання конкретних SQL-запитів по відношенню до бази даних. Для доступу до інформації в базі даних через конкретний запит визначення запитів звичайно використовується разом з наборами записів.

Об’єкти визначення таблиць (TableDef) представляють структуру таблиць в базі даних. За допомогою об’єктів визначення таблиць можна створювати нові таблиці та змінювати структуру та характеристики існуючих.

Існують ще декілька типів об’єктів DAO. Вони (Field, Parametr, Index, User, Group, Error) не представлені конкретними класами MFS. Об’єкти DAO цього типу доступні через інші відповідні класи DAO MFS.

Об’єкти **CDaoRecordset** представляють набори записів. Такий набор може представляти записи в таблиці, динамічний і статичний набори. Набір записів типу таблиці є поновлюваним і представляє записи в одній таблиці. Набір записів динамічного типу представляє записи з одної або декількох таблиць результатом запиту. Записи динамічного типу також поновлювані. Статичній набір може містити поля з одної або декількох таблиць, але ці поля не поновлювані. Статичний набір – це статична копія записів, що використовується для пошуку даних або створення звітів.

Набір записів представляє велику кількість функцій. Найважливішими з них є функції переміщення по набору записів та функції поновлення даних. К функціям переміщення відносяться Find, FindFirst, FindLast, FindPrev, Move, MoveFirst, MoveLast, MoveNext, MovePrev. До числа функцій поновлення входять AddNew, CancelUpdate, Delete, Edit, Update.

Інші функції, пов’язані з переміщенням, - GetAbsolutePosition, GetBookmark, GetPercenrPosition, SetAbsolutePosition, SetBookmark, SetPercentPosition.

Клас CDaoRecordset представляє різні функції атрибутів для установки та отримання атрибутів набору записів. Напиклад, функцію CanUpdate можна використовувати для визначення, чи є набір записів поновлюваним; функція SetCurrеntIndex застосовується для установки поточного індексу на набір записів таблиці.

Звичайно клас CDaoRecordset використовується шляхом створення похідного від нього класу набору записів, додавання змінних класу, що представляють поля, та перекриттям функції DoFieldExchange для підтримки обміну даними між базой даних та змінними класу. Проте існує декілька методів, що забезпечують альтернативну можливість. До їх числа входять функції GetFieldValue та SetFieldValue, які представляють можливість прямого доступу до значення поля за ім’ям. Цей метод називають динамічним зв’язуванням, на противагу статичному, який виконується за допомогою DoFieldExchange.

Інші операції з наборами даних можна використовувати для керування локально підтримуємим кешем записів і для маніпулювання індексами набору записів.

Клас **CDaoDatabase** представляє з’єднання з базою даних. З’єднання відбувається викликом функції CDaoDatabase::Open і переривається викликом CDaoDatabase::Close. Нова база даних може бути створена за допомогою функції CDaoDatabase::Create.

Клас CDaoDatabase представляє функції атрибутів; наприклад, GetName можна використовувати для отримання імені бази даних, а IsOpen – для визначення, чи відкрито з’єднання, що представляється об’єктом CDaoDatabase.

Інші методи використовуються для маніпулювання наборами об’єктів визначення таблиць і визначення запитів для цієї таблиці даних. Зокрема, функцію DeleteTableDef використовують для видалення з бази даних не тільки об’єкта TableDef DAO, але і основної таблиці і всіх її даних.

 Клас **CDaoWorkspace** представляє сеанси баз даних. Як правило, не треба створювати об’єкти типу CDaoWorkspace, якщо немає необхідності використовувати спеціальні функціональні можливості, що доступні за допомогою цього класу, або отримувати доступ до баз даних, захищених паролем.

Робочій простір DAO може бути створений викликом функції CDaoWorkspace::Create. Аргументи цієї функції вказують ім’я робочого простору, ім’я користувача та пароль. Існуючий об’єкт робочого простору може бути відкритий викликом CdaoWorkspace::Open; робочий простір по замовченню може бути явно відкритий шляхом передачі цій функції параметра NULL.

Існує декілька функцій, які маніпулюють базами даних та самою Jet-машиною баз даних. Наприклад, базу даних можна зжати або відновити викликом функції CompacDatabase або RepairDatabase. Інші функції можна використовувати для маніпулювання іменами користувачів, паролями або іншими атрибутами баз даних.

 Клас **CDaoQueryDef** представляє визначення запитів. Для створення нового визначення запиту слід скористатися функцією CQueryDef::Create; для доступу до визначення запиту, збереженого в базі даних, необхідно використовувати функцію CQueryDef::Open. Щойно створений запит може бути доданий до бази даних викликом функції CQueryDef::Append.

Об’єкти CQueryDef використовуються разом з об’єктами CRecordset для отримання даних з бази даних, а також - безпосередньо; для виконання запиту на дію, яка міняє дані в базі даних, слід використовувати функцію CQueryDef::Execute.

Інші методи CQueryDef використовують для установки і отримання атрибутів визначення запитів і для маніпулювання полями запитів і параметрами.

Клас **CDaoTableDef** представляєвизначення таблиць. Визначення таблиці описує структуру і атрибути таблиці в базі даних.

Існуюче в базі даних визначення таблиці може бути відкрито викликом функції CDaoTableDef::Open. Нове визначення таблиці може бути створене викликом функції CDaoTableDef::Create. Для додавання до бази даних таблиці, що відповідає новому визначенню, слід викликати функцію Append.

Поля можуть створюватися і видалятися за допомогою функцій CreateField та DeleteField. Індекси для таблиці можуть бути створені або видалені викликом функцій CreateIndex та DeleteIndex. Інші функції можна використовувати для установки або отримання різних атрибутів таблиці, наприклад, GetFieldCount повертає кількість полів в таблиці, а SetValidationRule використовується для присвоєння полю умови перевірки на значення.

Крім цих основних класів DAO операції DAO використовують два додаткових класи: CDaoFieldExchange та CDaoException.

**CDaoFieldExchange** використовується при звертаннях до CDaoRecordset::DoFieldExchange. Об’єкт типу СdaoFieldExchange визначає поле, що змінюється операцією обміну полями, та забезпечує інші параметри, що характерізують обмін полями.

Всі класи DAO використовують об’єкти виключення типу CDaoException для повідомлення про помилки.

*Побудова застосування DAO.*

.Для побудови застосування DAO необхідно перш за все створити джерело даних. Нехай ним буде база даних, створена в Microsoft Access. База даних Employee складається з 2 таблиць, де міститься інформація про працівників (їх ім’я та вік), та відомості про робочі плани (назва плану та максимальний вік робітника, якому буде дозволено займатися цим планом).

Створення застосування-DAO здійснено за допомогою AppWizard. Проект – однодокументний, в якості джерела даних (Data Source) вибано DAO, а дійсним файлом бази даних – базу даних Employee.

Структура застосування, яка створюється AppWizard схожа на відповідну для ODBC.

Класи застосування:

Новий клас CDAODBSet (DAODB – назва створюваного проекту) породжений від CDaoRecordset і представляє набір рядків, які будуть вибиратися з об’єднання всіх таблиць. AppWizard вставляє змінні, які відповідають стовпцям (полям) таблиць.

Об’явлення класу CDAODBSet:

class CDAODBSet : public CDaoRecordset

{

public:

 CDAODBSet(CDaoDatabase\* pDatabase = NULL);

 DECLARE\_DYNAMIC(CDAODBSet)

// Field/Param Data

 //{{AFX\_FIELD(CDAODBSet, CDaoRecordset)

 Cstring m\_LastName;

 Cstring m\_FirstName;

 long m\_Age;

 Cstring m\_Name;

 long m\_MaxAge;

 //}}AFX\_FIELD

// Overrides

 // ClassWizard generated virtual function overrides

 //{{AFX\_VIRTUAL(CDAODBSet)

 public:

 virtual Cstring GetDefaultDBName(); // REVIEW: Get a comment here

 virtual Cstring GetDefaultSQL(); // default SQL for Recordset

 virtual void DoFieldExchange(CDaoFieldExchange\* pFX); // RFX support

 //}}AFX\_VIRTUAL

// Implementation

#ifdef \_DEBUG

 virtual void AssertValid() const;

 virtual void Dump(CdumpContext& dc) const;

#endif

};

Реалізація класу CDAODBSet показує, як ці змінні ініціалізуються в конструкторі класу. Звертання до цих змінних є також в реалізації функції DoFieldExchange, сгенерованої мастером AppWizard. Ця функція здійснює обмін даними між змінними-членами в класі і полями в базі даних.

Реалізація класу CDAODBSet:

IMPLEMENT\_DYNAMIC(CDAODBSet, CDaoRecordset)

CDAODBSet::CDAODBSet(CDaoDatabase\* pdb)

 : CDaoRecordset(pdb)

{

 //{{AFX\_FIELD\_INIT(CDAODBSet)

 m\_LastName = \_T(“”);

 m\_FirstName = \_T(“”);

 m\_Age = 0;

 m\_Name = \_T(“”);

 m\_MaxAge = 0;

 m\_nFields = 5;

 //}}AFX\_FIELD\_INIT

 m\_nDefaultType = dbOpenDynaset;

}

Cstring CDAODBSet::GetDefaultDBName()

{

 return \_T(“D:\\Visual Studio\\MyProjects\\DAODB\\Employee1.mdb”);

}

Cstring CDAODBSet::GetDefaultSQL()

{

 return \_T(“[Employees],[Plans]”);

}

void CDAODBSet::DoFieldExchange(CDaoFieldExchange\* pFX)

{

 //{{AFX\_FIELD\_MAP(CDAODBSet)

 pFX->SetFieldType(CDaoFieldExchange::outputColumn);

 DFX\_Text(pFX, \_T(“[LastName]”), m\_LastName);

 DFX\_Text(pFX, \_T(“[FirstName]”), m\_FirstName);

 DFX\_Long(pFX, \_T(“[Age]”), m\_Age);

 DFX\_Text(pFX, \_T(“[Name]”), m\_Name);

 DFX\_Long(pFX, \_T(“[MaxAge]”), m\_MaxAge);

 //}}AFX\_FIELD\_MAP

}

Для виконання задачі DoFieldExchange використовує функції DFX\_. Ці функції є DAO-аналогами функцій RFX\_, що використовуються для обміну полями ODBC.

*ФункціїDFX\_.*

|  |  |  |
| --- | --- | --- |
| Ім’я функції | Тип поля | Тип ODBC SQL |
| FX\_Binary | CByteArray | DAO\_BYTES |
| DFX\_Bool | BOOL | DAO\_BOOL |
| DFX\_Byte | BYTE | DAO\_BYTES |
| DFX\_Currency | COleCurrency | DAO\_CURENCY |
| DFX\_DateTime | COleDateTime | DAO\_DATE |
| DFX\_Double | double | DAO\_R8 |
| DFX\_Long | long | DAO\_I4 |
| DFX\_LongBinary | CLongBinary | DAO\_BYTES |
| DFX\_SHORT |  |  |
| short |  |  |
| DAO\_I2 |  |  |
| DFX\_Single |  |  |
| float |  |  |
| DAO\_R4 |  |  |
| DFX\_Text |  |  |
| Cstring |  |  |
| DAO\_CHAR,DAO\_WCHAR |  |  |

Клас документа CDAODBDoc містить нову змінну m\_dAODBSet типу CDAODBSet. Ця змінна представляє набір записів, з якими пов’язаний документ. Клас перегляду CDAODBView містить вказівник типу CDAODBSet (m\_pSet); в реалізації по замовченю він встановлений для вказівки на змінну m\_dAODBSet об’єкта-документу. Клас перегляду також містить нову функцію OnGetrecordset, яка в реалізації по замовченню повертає m\_pSet.

**Настройка застосування.**

В діалогове вікно IDD\_DAODB\_FORM додано відповідні статичні елементи керування та елементи керування редагуванням. За допомогою діалогового вікна Add Member Variable майстера ClassWizard ідентифікуємо змінні діалогового вікна відповідними змінними наборами записів. Це робиться за допомогою відповідної змінної m\_pSet.

Для зміни критерія вибору можна скористатися функцією CDAODBSet::GetDefaultSQL. Реалізація цієї функції по замовченню повертає імена імена таблиць, з яких вибираються записи. В SQL потрібний нам вибір можна було б реалізувати:

**SELECT Employees.LastName, Employees.FirstName, Employess.Age, Plans.Name, Plans.MaxAge**

**FROM Employees, Plans**

**Where Employees.Age<Plans.MaxAge**

**ORDER BY Employees.LastName, Employees.FirstName, Plans.Name**

Вибрати відомості про прізвища та імена працівників, а також назви планів, для тих працівників, вік яких не перевищує максимальний вік, допустимий для цього плану.

При цьому вивід повинен бути відсортований спочатку за прізвищами працівників, потім за їх іменами, і накінець, за назвами планів.

Для ствоення відповідного запиту можна використовувати змінні класу CDaoRecordset. Цей клас пропонує дві змінні, одна з яких (m\_strFilter) відповідає умові SQL **WHERE** , а інша (m\_strSort) – умові SQL **ORDER BY**.

Оновлення CDAODBSet::GetDefaultSQL

**Cstring CDAODBSet::GetDefaultSQL()**

**{**

 **m\_strFilter=\_T(“[Employees].[Age]<[Plans].[MaxAge]”);**

 **m\_strSort=\_T(“[Employees].[LastName],[Employees].[FirstName],[Plans].[Name]”);**

 **return \_T(“[Employees],[Plans]”);**

**}**

Застосування має такий вигляд:

За допомогою Component Galery до проекта було ще додано екранну заставку (Splash Screen)



Та системна інформація для діалогового вікна About (System Info for About Dialog).

**Візуальні інстументи бази даних.**

Visual C++ інтегрує в Visual Studio ряд інструментів, пов’язаних з базами даних. Тепер можна створювати бази даних, заповнювати таблиці, створювати і тестувати запити та відлагоджувати збережені процедури на віддаленому або локальному сервері, не залишаючи середовище Visual Studio.

Багато з можливостей розробки баз даних доступні для більшості джерел даних, інші, такі як створення таблиці, характені для Microsoft SQL Server.

Дві можливості представляють особливий інтерес. Перша – Query Designer, який дозволяє створювати і тестувати запити SQL з Visual Studio: друга – відладчик SQL, який дозволяє відлагоджувати збережені процедури, що виконуються на Microsoft SQL Server.

Для демонстації вищезазначених можливостей була створена база даних за допомогою Microsoft Access. Ця база містить дві таблиці: таблиця Books відображає книги з їх назвами, авторами та ISBN індексами; таблиця Authors містить відомості про авторів. Обидві таблиці з’єднуються відношенням – “багато-до-багатьох”: Books.Authors -> Authors.Name.

Для доступу до бази даних з Visual Studio необхідно створити проект бази даних (Database project). Доступ до джерела даних здійснюється через вказівку імені файлу джерела даних. Це файл – Books.**DSN**.

У вкладці FileView у вікні Workspace є лише один файл – файлове джерело даних (файл DSN):



У вкладці DataView можна побачити таблиці і поля, а також запити, якщо вони були створені:



Таким чином, тепер можна перевіряти і змінювати зміст таблиць, створювати SQL запити. Не можна створювати нові таблиці.

Для додавання або зміни даних достатньо двічі натиснути на відповідній таблиці у вкладці DataView. При цьому також з’являється вікно QUERY.

**Створення запитів.**

Побудуємо запит: **вибрати всі книги, написані американськими авторами.**

New Query можна створити натиснувши на імені бази даних у вкладці FileView:



Вибір таблиці здійснюється перетаскуванням відповідного імені таблиці з вікна DataView в вікно перегляду діаграм вікна Query. Встановлення з’єднання між полями – аналогічне (просте перетаскування):



Критерій вибору відображається в панелі сітки:



Панель SQL відображає відповідний SQL-запит:



Панель результатів відображає результати запитів:



Можна створювати не тільки запити-SELECT, а й INSERT, UPDATE, DELETE.

Запит UPDATE не має завершеного набору; при виконанні цього запиту Visual Studio в діалоговому вікні відображає тільки кількість записів, що були модифіковані:

**Висновки.**

 В даній роботі, за допомогою створення власних прикладів, були розглянуті основні можливості програмування баз даних в Visual C++, зокрема використовуючи ODBC та DAO, а також можливості візуального конструювання баз даних.

***Резюме:***

**ODBC** – могутній, незалежний від постачальника механізм доступу до інформації з різних джерел даних.

* В основі ODBC лежать драйвери ODBC.
* Джерелами даних можуть бути локальні файли та сервери віддалених даних.
* Джерела даних звичайно вказуються за допомогою програми установки ODBC, хоча функція SQLDriverConnect робить можливим підключення до джерела даних, який і не був встановлений таким чином.
* Сеанс ODBC містить звертання до функцій, які будують з’єднання з джерелом даних, конструюють і представляють оператори SQL та обробляють результати.
* ODBC підтримує варіанти стандартного SQL – оператори маніпулювання даними (SELECT, INSERT, UPDATE, DELETE) та оператори визначення даних (CREATE TABLE, DROP TABLE, CREATE INDEX, DROP INDEX, ALTER TABLE, CREATE VIEW).
* MFS надає два класи для підтримки ODBC – CDatabase та CRecordset.

**DAO** - представляє засновану на OLE технологію, що використовується в Visual Basic for Application та Microsoft Access для доступу до баз даних за допомогою Jet-машини баз даних Microsoft.

* MFS, Visual C++ AppWizard та ClassWizard забезпечують підтримку розробки DAO-застосувань в Visual C++.
* Бібліотеки DAO поставляються у вигляді компонент.
* Для спрощення представлення DAO пропонується 5 основних (CDaoRecordset, CDaoDatabase, CDaoWorkspace, CDaoQueryDef, CDaoTableDef) та 2 додаткових класи (CDaoFieldExchange, CDaoException).

Visual C++ пропонує **візуальні інструменти** для роботи з базами даних.

* Деякі з цих інструментів доступні для більшості джерел даних ODBC (створення проекту Database project); інші характерні для Microsoft SQL Server (створення проекту New Database Wizard).
* Проект бази даних може бути створений для доступу до інформації, що міститься в існуючій базі даних. Дані можна редагувати, додавати та видаляти; можна створювати запити.
* При наявності Microsoft SQL Server доступний ряд додаткових можливостей, зокрема можна додавати таблиці та, використовуючи діаграми баз даних, можна вказувати взаємозв’язки.

**Список створених проектів:**

В ході написання даної курсової роботи були створені такі програмні продукти:

1. Консольне застосування - **ODBC\_Excel** – програма ODBC, що може бути відкомпільована як в самому середовищі Visual C++, так і з командного рядка компілятором *cl.exe*. Дане застосування ілюструє можливості використання функції SQLDriverConnect, яка надає можливість підключення до джерела даних, навіть якщо той не був раніше інстальований за допомогою програми початкової установки ODBC. Джерелом даних для цього прикладу є таблиця *Student*, створена в Microsoft Excel. Також ілюструються можливості інших функцій, пов’язаних з створенням SQL-запитів. Програма відображає виконання двох запитів
2. Проект - **MFSDB** – застосування, що використовує технологію ODBC, створене майстером AppWizard. Використання ODBC в цьому випадку спрощується за допомогою Microsoft Foundation Classes Library (MFS). Джерелом даних, що було інстальовано за допомогою початкової установки ODBC, є база даних *Students*, створена в Microsoft Access. Розглянуті можливості створення запитів безпосередньо в середовищі Visual C++, а саме використовуючи функцію GetDefaultSQL класу, породженого від CRecordset. Створена форма, яка відображає результати запиту. В застосуванні також створені додаткові класи, породжені від CRecordset, що дають змогу писати додаткові запити або використовувати інші таблиці з даної бази даних. Створено декілька форм, класи яких породжені від CRecordView. Одні з них відображають результати роботи запитів, написаних за допомогою вищезазначеної функції, інші використовують запит, створений в Microsoft Access. Розглянуті можливості використання Component Galery (до проекту додано компоненту Tips of the Day – Поради дня).
3. Проект - **DAODB** - застосування, що використовує технологію DAO, створене майстером AppWizard. Джерелом даних є база даних *Employees*, створена в Microft Access. В застосуванні створена форма, яка відображає результати запиту, написаного за допомогою функції GetDefaultSQL. До проекту додані компоненти з Component Galery - Splash Screen (Екранна заставка) та System Info for About Dialog (Системна інформація длі діалогового вікна About).
4. Проект **Book** – проект Database project, ілюструє можливості бази даних *Books*, створеної в Microsoft Access, в Visual Studio. Таблиці бази даних можна передивлятися, редагувати, додавати нові записи чи видаляти існуючі. Можна створювати нові запити, запускати їх на виконання, відлагоджувати, і все це не залишаючи середовища Visual C++.

*Список використаної літератури.*

1. В.Тосс и др. Visual C++5. Энциклопедия пользователя. -Киев: DiaSoft, 1998.-683c.
2. Д.Беннет. Visual C++5. Руководство разработчика. -Киев: Диалектика, 1998.-760с.
3. Джон Вейскас. Эфективная работа с Access 7.0. -Санкт-Петербург: Питер, 1997.-830с.
4. К. Дейт. Введение в систему баз данных. -М.: Наука,1988.-463с.
5. Дж. Вебер. Технология Java в подлиннике. -Санкт-Петербург: BHV, 1998.-1070с.