

УДК 681.3.06

С. О. Спасітелєва

ЗАСОБИ ДОСТУПУ ДО БАЗ ДАНИХ У ОБ'ЄКТНИХ МОВАХ ПРОГРАМУВАННЯ

Стаття присвячена процесу сумісного використання об'єктно-орієнтованих мов програмування та реляційних баз даних. Досліджуються різноманітні технології доступу до баз даних, які використовуються мовами C++ та Java при розробці прикладних систем. Проаналізовані можливості технологій доступу до різних джерел даних, їх недоліки та переваги.

З широким використанням мережевих технологій, Інтернет/Інтранет виникає потреба у створенні нових прикладних систем (ПС) для роботи з даними, аналізу даних, підтримки бізнес-процесів. При цьому розробники здебільшого змушені використовувати існуючі реляційні бази даних (БД). При розробці ПС означеного вище типу виникає потреба у вирішенні проблеми доступу до таблиць реляційних БД різних виробників, а також об'єднання даних із таких таблиць і їх спільного використання. Аналізу підходів до вирішення такої типової задачі і присвячена ця стаття, в якій розглядаються питання сумісного використання реляційних БД різних виробників та об'єктних мов програмування. У зв'язку з широким розповсюдженням мов програмування C++, Java та їх використанням для розробки ПС аналіз технологій доступу до БД проводиться з використанням цих мов.

Доступ до баз даних з використанням мови Java. Для доступу до БД серед програм, написаних мовою Java, можна запропонувати спільне використання інтерфейсу JDBC та специфікацій SQLJ [1].

Інтерфейс JDBC, перша версія якого входила до складу JDK 1.1, визначає Java API для доступу до реляційної СКБД. JDBC API функціонально досить багатий. Він забезпечує класи і методи для того, щоб з'єднуватися з базою даних; отримувати метадані з БД; виконувати запит або оператор DDL; готувати оператор DML або оператор CALL з параметрами, які будуть задані під час виконання оператора; здійснювати вибірку як даних, так і метаданих для результатуючих наборів, породжених виконанням оператора. JDBC забезпечує динамічне виконання операторів SQL. При будь-яких синтаксичних

або семантичних помилках в операторах SQL будуть виникати виняткові ситуації під час виконання додатку. Драйвер JDBC [2] підтримує оператори базового рівня (Entry Level) SQL-92 з деякими розширеннями, визначеними у специфікації JDBC. Він може також підтримувати оператори інших рівнів SQL і оператори, які є розширеннями постачальника до стандарту SQL. SQLJ [3] дозволяє вбудовувати статичні оператори SQL в програми мови Java значною мірою таким же чином, як SQL-92 дозволяє вбудовувати оператори SQL у Сі, КОБОЛ і деякі інші мови. SQLJ підтримується засобами JDBC. Динамічні оператори SQL підтримуються JDBC. Транслятор SQLJ відшукуватиме вбудовані оператори SQL і замінюватиме їх операторами Java, які забезпечують виконання відповідних операторів SQL. Отримана в результаті початкова програма на мові Java буде компілюватися звичайним чином. Для виділення операторів мови SQL використовуються відповідні ідентифікатори. Транслятору SQLJ може бути наказано встановити з'єднання з примірником БД під час його роботи і використовувати метадані, які він там знайде, для перевірки правильності операторів SQL.

Наступний фрагмент коду показує, як SQLJ може використовуватися для доступу до БД. Нехай існує таблиця employee (службовці) і в ній стовпець emp_id (ідентифікатор службовця), тоді:

```
try {#sql { DELETE FROM employee
WHERE emp_id = 17 }; }
catch (SQLException sqe) {System.out.
println
(sqe.getMessage());},
```

де `#sql {...}`; ідентифікує виконавчий оператор SQL. Фігурні дужки `{}` використовуються як обмежувачі цього оператора SQL, і вони відокремлюють його від іншої частини програми мовою Java. У наведеному прикладі не показані деякі елементи настройки, необхідні для використання цього прикладу (драйвер JDBC повинен бути зареєстрований менеджером драйверів JDBC, необхідно здійснити з'єднання з БД).

Для ілюстрації розглянемо деякі можливості SQLJ. Безсумнівно оператор `SELECT` найчастіше використовується SQL в додатках. Для доступу до рядків результату обробки запиту використовується ітератор результуючого набору, який можна приблизно порівняти з курсором SQL. Оскільки ітератор результуючого набору – об'єкт Java в SQLJ, він може бути переданий як аргумент у виклику метода. SQLJ забезпечує два типи ітераторів результуючих наборів. Ці два типи не можуть змішуватися для одного результуючого набору. Повинен бути вибраний тільки один із них.

Зв'язування зі стовпцями за іменем. Перший з цих типів, іменований ітератор, представлений таким прикладом:

```
#sql iterator Employee
(int emp_id, String emp_lname,
 java.sql.Date start date);
Employee emp;
#sql emp = { SELECT emp_lname,
emp_id, start_date FROM employee
WHERE emp_fname LIKE C% };
while (emp.next()) { System.out.
println emp.start_date() + "," +
emp.emp_id() + "," +
emp.emp_lname().trim()); }
emp.close();
```

Оператор ітератора `"#sql iterator..."` в SQLJ визначає клас `Employee` з методами доступу для кожного із стовпців результату виконання запиту. Генеруються також методи, подібні `next()` і `close()`. Створюється і зв'язується з результатом виконання запиту об'єкт `emp`. Далі цикл `while` повторюється для рядків результату запиту і друкує кожний з них. Наприкінці результат запиту закривається. Саме таке використання пар "ім'я стовпця/тип даних" в описі ітератора визначає, що це – іменований ітератор. Імена стовпців відповідають іменам в ітераторі без урахування регістра (case-insensitive).

Зв'язування зі стовпцями за позицією. Другий тип ітератора результуючого набору, що використовується для звернення до результату виконання запиту, – позиційований ітератор, який ілюструється таким прикладом:

```
#sql iterator Employee (int, String, String);
int emp_id = 0;
String emp_lname = null; String
emp_fname = null;
Employee emp;
#sql emp = { SELECT emp_id, emp_lname,
emp_fname FROM
employee WHERE emp_fname LIKE C% };
while (true) { #sql {FETCH: emp INTO:
emp_id, :emp_lname, :emp_fname};
if (emp.endFetch()) break;
System.out.println(emp_fname.trim() +
" " + emp_lname.trim() + ", " + emp_id ); }
emp.close();
```

Оператор `iterator` в цьому прикладі специфікує тільки типи даних, а не пари "ім'я стовпця/тип даних", вказуючи тим самим, що був вибраний варіант позиційованого ітератора. Під час виконання оператора `FETCH` стовпці результату запам'ятовуються в Java-змінних у тому порядку, в якому вони були визначені. Клас `Employee` має методи такого роду, як `endFetch()` і `close()`. Додаток сканує рядки результату виконання запиту за допомогою оператора `FETCH` мови SQL.

Вибір між іменованим і позиційованим ітераторами є повністю стилістичним. Для використання іменованого ітератора всі стовпці цільового списку оператора `SELECT` повинні мати унікальні імена або ними повинні бути присвоєні унікальні імена шляхом використання можливості SQL з перейменування стовпців (фраза `AS`).

Висновки щодо використання SQLJ. Наведені приклади ілюструють загальний підхід до використання SQLJ і не претендують на повноту викладення усіх можливостей роботи з даними. Спираючись на опис специфікацій та досвід роботи, можна зробити такі висновки щодо створення Java-програм з доступом до БД. Java-програми не залежать від апаратної платформи, на якій вони виконуються. Оскільки програми SQLJ перетворюються на чистий код Java, вони можуть виконуватися всюди, де існує віртуальна машина Java (JVM). Подібним чином JDBC дозволяє розробляти додатки, незалежні від СКБД, яка буде використовуватися під час виконання. Оскільки SQLJ генерує звертання до JDBC, один додаток SQLJ може виконуватися з багатьма СКБД.

SQLJ забезпечує додатковий рівень незалежності від СКБД. Оператори SQL, які були оброблені засобами SQLJ, можуть далі оброблятися настроявачами (customizer), орієнтованими на продукти різних постачальників. Цей настроявач може, наприклад, генерувати код для створення і виконання процедур, які містять початкові опера-

тори додатка SQL. Такий згенерований код стане частиною додатку SQLJ. Якщо під час виконання існує настройка для СКБД, з якою здійснюється з'єднання, то буде використовуватися код, згенерований відповідним настроювачем. Якщо ніякої такої настройки не знайдено, буде використовуватися початковий код JDBC.

Доступ до баз даних з використанням мови C++. IDE Visual C++ пропонує розробникам різні технології для доступу до даних та маніпулювання даними. Технології підтримуються множиною класів із бібліотеки MFC (Microsoft Foundation Classes). При розробці прикладних систем можна використовувати такі технології доступу до БД:

- ODBC (Open Database Connectivity) – зв'язок відкритих баз даних. Забезпечує незалежний від постачальника механізм доступу до даних з різних джерел. Архітектура ODBC підтримується системою драйверів для спільного доступу до різноманітних баз даних, що дає можливість додаткам зв'язуватися з БД, використовуючи загальний програмний інтерфейс доступу до даних.

- DAO (Data Access Objects) – об'єкти доступу до даних. Забезпечує можливість доступу і маніпулювання базами даних за допомогою машини баз даних Microsoft Jet. DAO базується на технології OLE.

- OLE DB – технологія зв'язування і впровадження об'єктів баз даних. Являє собою інтерфейс БД оснований на моделі Common Object Model (COM), дозволяє одержувати доступ до широкого спектра різноманітних типів даних. Фактично OLE DB – це набір специфікацій інтерфейсів компонентної об'єктної моделі, що об'єднує традиційні реляційні БД і інші джерела даних, – від простих файлових систем до електронної пошти.

- ADO (ActiveX Data Objects) – об'єкти даних ActiveX. Являють собою множину об'єктів COM. Технологія спеціально розроблена для додатків, призначених для доступу і маніпулювання даними в БД. ADO є високошвидкісною компонентною технологією, що добре підходить для розробки матеріалів, орієнтованих на Web.

Бібліотека MFC забезпечує розробників PC множиною класів ODBC та DAO. Класи використовують прикладний програмний інтерфейс (API) ODBC та DAO і спрощують їх використання в додатках C++.

Використання ODBC значною мірою спрощується завдяки MFC. Набір класів, що надають-

ся MFC для підтримки додатків ODBC [4], представлено на *Рис. 1*. Найголовніші класи, що надаються MFC для підтримки застосувань ODBC, – це класи CDatabase та CRecordset. Клас

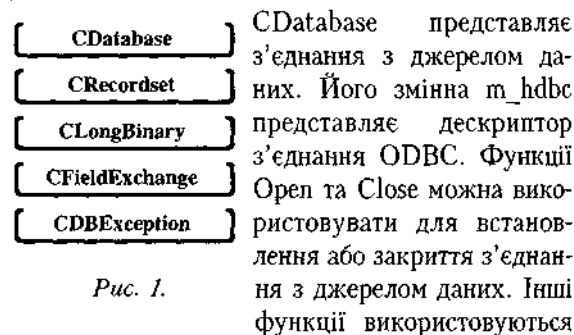


Рис. 1.

для встановлення або отримання параметрів з'єднання. За замовчуванням для доступу до джерела даних клас CDatabase використовує асинхронний режим. Виконувана асинхронна дія може бути перервана викликом функції Cancel.

Обробка транзакцій підтримується функціями BeginTrans та Rollback. Функція OnSetOptions використовується для встановлення стандартних опцій з'єднання. Функцію ExecuteSQL можна використовувати для безпосереднього виконання оператора SQL.

Клас CRecordset використовується для інкапсуляції запитів до БД, таких як додавання, модифікація і вилучення рядків. Змінні-члени цього класу ідентифікують дескриптор оператора ODBC, число полів і параметрів у наборі даних, об'єкт CDatabase, за допомогою якого набір записів є підключеним до джерела даних, і два рядки, що відповідають умовам WHERE та ORDER BY.

Коли набір записів відкривається викликом функції Open, відбувається доступ до таблиці і виконується запит, що зображується набором записів. 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. Переміщуватися по набору записів можна за допомогою функцій Move, MoveFirst, MoveLast, MoveNext та MovePrev. Дії з набором записів можуть виконуватися 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 полям таблиці, яку зображують набір записів. Потім треба оновити функцію DoFieldExchange набору записів; ця функція за

допомогою функцій обміну полями записів RFX_ (Record Field Exchange) повинні сприяти обміну даних між змінними класу та полями у БД. Ці функції за синтаксисом аналогічні функціям обміну даними діалогу DDX_ (dialog data exchange). Обмін полями підтримується за допомогою класу CFieldExchange. Об'єкт цього класу містить інформацію про поля, обмін якими повинен відбуватися при викликанні функції DoFieldExchange набору записів.

Клас CRecordset є похідним класом від CFormView, який розроблено спеціально для відображення в формах записів БД. Об'єкти типу CRecordView використовують функції обміну даними діалогу та обміну полями записів для підтримки переміщення даних між формою і джерелом даних. Об'єкти, похідні від CRecordView використовуються разом з об'єктами, похідними від CRecordset.

Для ілюстрації наведемо приклад роботи з БД Student, яка складається з трьох таблиць, що містять інформацію про студента, предметів, що вивчає студент, та оцінки з цих предметів. Якщо створити в додатку новий клас CMFCODBCSet, який є похідним від CRecordset, тоді при оголошенні цього класу необхідно додати змінні-члени, що відображають поля вказаної таблиці БД. Ці змінні також відображаються у файлі реалізації класу, в конструкторі, а також у функції DoFieldExchange. Остання викликається MFC для обміну даними між змінними набору записів і відповідними стовпцями в таблиці БД.

Обмін інформацією між користувачем та додатком зручно здійснювати через діалогове вікно з відповідними елементами керування і скористатися майстром ClassWizard для додавання відповідних змінних класів.

До діалогового вікна можна додати статичні елементи керування та елементи редагування, що будуть відображати необхідну інформацію, вибірка якої буде здійснюватися запитом для віднайдення усіх студентів, що мають академічну заборгованість:

```
SELECT DISTINCT student.Name, sub-
ject.Subject
FROM (Mark INNER JOIN student ON
Mark.Name = student.Name) INNER JOIN sub-
ject ON Mark.Subject = subject.Subject
WHERE (Mark.Mark)<61);
```

Даний запит можна реалізувати безпосередньо у середовищі Visual C++ за допомогою функції CRecordset::GetDefaultSQL. Її змінна m_strFilter відповідає умові SQL WHERE.

```
Cstring CMFCODBCSet::GetDefaultSQL()
{m_strFilter=_T("[Mark].[Mark]<61");
return _T("[Mark],[student]");}
```

Використання DAO спрощується завдяки MFC. Набір класів для підтримки DAO [4] із MFC зображено на Рис.2. Існує 5 основних та 2 допоміжних класи, пов'язаних з DAO. Всі об'єкти DAO похідні від DBEngine

Об'єкти БД (Database) і набору записів (Recordset) досить наочно представляють бази даних і набори вибірок (таблиці, набори записів або динамічні набори) у цих БД.

Об'єкти визначення запитів (QueryDef) використовуються для виконання конкретних SQL-запитів по відношенню до БД.

Об'єкти визначення таблиць (TableDef) представляють структуру таблиць у базі даних. За допомогою об'єктів визначення таблиць

можна створювати нові таблиці та змінювати структуру й характеристики існуючих.

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

Об'єкти CDaoRecordset представляють набори записів. Такий набір може представляти запис у таблиці, динамічний і статичний набори. Для роботи з набором записів можна використовувати багато функцій. Найважливішими з них є функції переміщення по набору записів та функції поновлення даних.

Клас CDaoRecordset представляє різні функції атрибутів для встановлення та отримання атрибутів набору записів. Цей клас CDaoRecordset використовується шляхом створення похідного класу набору записів, додавання змінних класу, що представляють поля, та перекриттям функції DoFieldExchange для підтримки обміну даними між базою даних та змінними класу.

Клас CDaoDatabase забезпечує з'єднання з БД. З'єднання відбувається викликом функції CDaoDatabase::Open і переривається викликом CDaoDatabase::Close. Нова БД може бути створена за допомогою функції CDaoDatabase::Create.

Інші методи використовуються для маніпулювання наборами об'єктів визначення таблиць і визначення запитів для цієї таблиці даних.

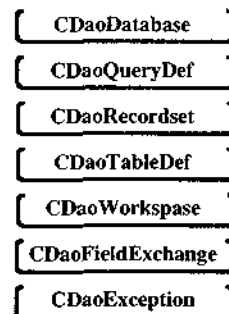


Рис. 2.

Клас `CDaoWorkspace` представляє сеанси баз даних. Як правило, не треба створювати об'єкти типу `CDaoWorkspace`, якщо немає необхідності використовувати спеціальні функціональні можливості, доступ до яких отримується за допомогою цього класу, або отримувати доступ до БД, захищених паролем.

Робочий 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 за замовчуванням може бути явно відкритий шляхом передачі цій функції параметра `NULL`.

Клас `CDaoQueryDef` представляє визначення запитів. Для створення нового визначення запиту слід скористатися функцією `CQueryDef::Create`; для доступу до визначення запиту, збереженого у БД, 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 із БД, а також безпосередньо. Для виконання запиту на дію, яка змінює дані у БД, слід використовувати функцію `CQueryDef::Execute`. Інші методи `CQueryDef` використовують для встановлення і отримання атрибутів визначення запитів і для маніпулювання полями запитів і параметрами.

Клас `CDaoTableDef` представляє визначення таблиць. Визначення таблиці описує структуру і атрибути таблиці у БД.

Для ілюстрації наведемо приклад роботи з БД `Employees`, яка складається з двох таблиць, де міститься інформація про працівників (ім'я та вік), та про робочі плани (назва плану та максимальний вік працівника, якому буде дозволено займатися виконанням цього плану). Якщо створити в додатку новий клас `CMFCDAOSet`, який є похідним від `CdaoRecordse`, тоді при оголошенні цього класу необхідно додати змінні-члени, що відображають поля таблиць БД. Реалізація класу `CMFCDAOSet` показує, як ці змінні ініціалізуються у конструкторі класу. Ця функція здійснює обмін даними між змінними-членами у класі і полями у БД. Реалізація класу `CMFCDAOSet`:

```
IMPLEMENT_DYNAMIC(CMFCDAOSet,
CDaoRecordset)
CMFCDAOSet:: CMFCDAOSet (CDao-
Database* pdb) : CDaoRecordset(pdb)
{ //{{AFX_FIELD_INIT(CMFCDAOSet)
```

```
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
MFCDAOSet::GetDefaultDBName()
{return _T("D:\\Visual Studio\\
Employees.mdb");}
CString CMFCDAOSet::GetDefaultSQL()
{return _T("[Employees],[Plans]");}
void CMFCDAOSet::DoFieldExchange(CDa
oFieldExchange* pFX)
{ //{{AFX_FIELD_MAP(CDAODataSet)
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 зміни критерію вибору можна скористатися функцією `CDAODataSet::GetDefaultSQL`. Реалізація цієї функції за замовчуванням повертає імена таблиць, з яких вибираються записи. У SQL потрібний нам вибір можна реалізувати так:

```
SELECT Employees.LastName,
Employees.FirstName, Employees.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 максимального встановленого віку. При цьому виведення повинно бути відсортоване спочатку за прізвищами працівників, потім за їх іменами, і наприкінці, за назвами планів.

Для створення 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`.

```

Cstring CDAODataSet::GetDefaultSQL()
{m_strFilter=_T("[Employees].[Age]<[Plans].[MaxAge]");
 m_strSort=_T("[Employees].[LastName],[Employees].[FirstName],[Plans].[Name]");
 return _T("[Employees],[Plans]"); }

```

Висновки щодо застосування ODBC та DAO. Безперечно, найпоширенішими і достатньо зручними технологіями є ODBC та DAO. Різні аспекти використання означених вище технологій при створенні програм для роботи з даними застосовувались студентами департаменту комп'ютерних технологій НаУКМА. Це дозволило автору проаналізувати практичні аспекти використання даних технологій. ODBC та DAO мають деякі загальні можливості. Завдяки цьому Visual C++ дозволяє використовувати безпосередньо дві технології разом. Тобто, можна використовувати можливості, які надають як ODBC, так і DAO. Обидві технології спираються на об'єкти баз даних та набір записів з аналогічними компонентами. Код для програмування як ODBC, так і DAO значною мірою схожий.

Залежно від типу БД можна зробити чіткий вибір між ODBC та DAO. ODBC забезпечує загальний програмний інтерфейс доступу до БД від простих ASCII до більш складних баз даних – Oracle, SQL Server тощо. DAO надає можливість доступу і маніпулювання БД за допомогою машини баз даних Microsoft Jet, це накладає суттєві обмеження на вибір джерела даних, але пропонує додаткові можливості і спрощує використання. Наприклад, при застосуванні технології ODBC неможливо слідкувати за транзакціями з такою ж чіткістю, як за допомогою DAO. При використанні DAO з процесором Microsoft Jet користувач отримує підтримку для транзакцій на рівні робочої області, ODBC дає підтримку транзакцій тільки на рівні БД. Це уповільнює налагодження програм, а також може призвести і до інших проблем.

Класи DAO мають більше об'єктів та методів, ніж класи ODBC. Проте слід пам'ятати, що коли потрібний доступ до 16-розрядних БД, тоді не можна використовувати DAO.

Використання OLE DB. Новий програмний інтерфейс OLE DB для роботи з БД компанії Microsoft може замінити більш ранні версії програмних інтерфейсів DAO та Remote Access Objects [5]. OLE DB забезпечує гнучкий інтерфейс COM між провайдерами даних та споживачами даних. Провайдери даних (data providers) – це додатки, що містять свої власні дані і зображують їх як таблиці. Провайдери даних

реалізують інтерфейси COM OLE DB для групи рядків і можуть бути простими і представленими лише однією таблицею даних, або ускладненими розподіленими системами баз даних.

Споживачі (consumers) – це додатки, що використовують інтерфейси OLE DB для маніпулювання даними, які зберігаються у провайдера даних. Коротко розглянемо підходи до створення додатків-споживачів у OLE DB. Додатки-споживачі є тільки частиною архітектури OLE DB. OLE DB також дозволяє створювати власні провайдери даних, які підтримують власні типи даних, або провайдери служб, які вибирають дані у провайдерів даних та передають їх додаткам-споживачам.

Інтерфейс OLE DB складається з кількох класів об'єктів COM, що представляють різноманітні компоненти додатків баз даних. До них належать:

- *Переліки.* Використовуються для пошуку в системі доступних джерел даних та інших переліків, що можна використовувати рекурсивно. Досягається це шляхом використання інтерфейсу `ISourcesRowset`, що дозволяє генерувати групу рядків, яка описує доступні джерела даних. SDK (software development kit) OLE DB включає кореневий перелік, що знаходиться у реєстрі встановленого джерела даних та інші переліки.

- *Джерела даних.* Зображують окремі дані і провайдери служб. Вони використовуються для створення сеансів. Об'єкт джерела даних створюється з використанням методу `CoCreateInstance`. Джерело даних повинне реалізувати такі інтерфейси: `IDBProperties` – використовується для задання параметрів з'єднання і одержання властивостей провайдера; `IDBInitialize` – використовується для підключення до джерела даних; `IDBCreateSession` – використовується для з'єднання об'єктів сеансів для даного з'єднання; `IPersist` – використовується для збереження інформації з джерел даних у постійній пам'яті.

- *Сеанси.* Використовуються для створення транзакцій і команд. `IGetDataSource` використовується для визначення джерела даних, створеного сеансом. `IOpenRowset` використовується для відкриття групи рядків, що містить дані провайдера. `IsessionProperties` використовується для роботи з властивостями сеансу.

- *Транзакції.* Використовуються для групування кількох операцій у єдину транзакцію.

- *Команди.* Використовуються для передачі команд (наприклад, SQL) джерелу даних, одержання груп рядків. Екземпляри команд повинні

підтримувати такі інтерфейси. *Iaccessor* використовується для створення аксесорів (*accessor*), що визначають зв'язок параметрів команди або даних стовпчика з буферами даних. *IColumnInfo* дозволяє одержати інформацію про стовпчики в групі рядків. *ICommand* забезпечує методи виконання команди. *ICommandProperties* використовується для задання властивостей екземпляра команди. *ICommandText* використовується для задання тексту команди, що буде переданий провайдеру даних. *IConvertType* забезпечує інформацію про прийнятні типи перетворень.

- *Групи рядків*. Використовуються для роботи з табличними даними. Групи рядків створюються або об'єктами сеансів або екземплярами команд. Усі об'єкти групи рядків повинні реалізувати такі інтерфейси. *Iaccessor* використовується для створення аксесорів, що визначають зв'язок параметрів команди або даних стовпчика з буферами даних. *IColumnInfo* дозволяє одержати інформацію про стовпчики у групі рядків. *IConvertType* забезпечує інформацію про прийнятні типи перетворень. *IRowset* забезпечує інтерфейс для вилучення даних і керування рядками. *IRowsetInfo* використовується для роботи з властивостями групи рядків та іншої інформації про групу рядків.

- *Помилки*. Використовуються для одержання інформації про аварійні ситуації.

Архітектура OLE DB дозволяє використовувати різні типи даних та різних провайдерів служб. Найрозповсюдженішим провайдером даних є *MSDASQL*, який забезпечує доступ OLE DB до джерел даних ODBC. Цей провайдер дуже зручний, тому що має переваги існуючих драйверів ODBC та традиційних властивостей провайдерів OLE DB. Окрім того, до комплексу постачання *MSDASQL* входить SDK OLE DB.

Перед початком роботи з даними, запропонованими провайдером, необхідно створити та ініціалізувати об'єкт провайдера. Для того, щоб отримати можливість працювати з даними, пов'язаними з екземпляром джерела даних, необхідно створити об'єкт сеансу. Екземпляри сеансів використовуються для забезпечення контексту транзакцій, інтерфейсів для створення груп рядків та команд.

Для ілюстрації розглянемо приклад створення екземпляра команди OLE DB, який використовується для передачі тексту команди провайдеру. Найпоширенішими серед команд є оператори SQL. Новий екземпляр команди створюється за допомогою методу *IDBCreateCommand::CreateCommand()*. Для визначення тексту команди

викликається метод *ICommandText::SetCommandText()*. Для надсилання команди провайдеру необхідно викликати метод *ICommand::Execute()*, який виконує команду та повертає результат запиту. Наведений приклад показує, як можна створити команду, визначити текст команди та відправити її.

```
IDBCreateCommand* pIDBCreateCommand
= NULL;
ICommand* pICommand = NULL;
ICommandText * p ICommandText = NULL;
IRowset * pIComRowset = NULL;
hr = pSomeSessionInterface -
QueryInterface(
IID_IDBCreateCommand, (LPVOID*) &p
IDBCreateCommand);
hr = pIDBCreateCommand ->
CreateCommand (NULL,
IID ICommand, (IUnknown**)
&pICommand);
hr = pICommand ->
QueryInterface(IID ICommandText,
(LPVOID*) &pICommandText);
hr = pICommandText -> SetCommandText
(DEBUID_DBSQL,
L"SELECT * FROM MyTable");
hr = pICommand -> Execute (NULL,
IID IRowset, NULL, NULL,
(IUnknown**) &pIComRowset);
if (FAILED(hr))
TRACE ("Помилка виконання програми
\n");
```

Використання ADO. ADO є сучасним інтерфейсом для роботи з БД. В ньому міститься опис об'єктів, що можна використовувати для роботи з даними різноманітних типів додатків. ADO спирається на інтерфейс COM, що містить об'єкти, доступні для широкого спектра мов програмування, включаючи Visual C++, Visual Basic, VBScript і JavaScript. ADO можна використовувати в серверних додатках або додатках проміжного типу, особливо при роботі Active Server Pages компанії Microsoft.

Компанія Microsoft пропонує реалізацію ADO для доступу до будь-яких джерел даних OLE DB, включаючи новий провайдер Active Directory, який реалізує інтерфейс OLE DB для роботи з файловими системами. Ця реалізація ADO для OLE DB називається *ADODB*. *ADODB* може використовуватися для доступу до провайдера Microsoft OLE DB (*MSDASQL*), який забезпечує доступ до будь-яких джерел даних ODBC.

Об'єкти, які використовує ADO, простіші за об'єкти OLE DB. ADO визначає набір із семи типів об'єктів і чотирьох типів колекцій [5]. Усі вони зображені на Рис. 3. Ці об'єкти можна використовувати у додатках C++, які значно спростилися з появою підтримки нових класів COM. За допомогою ADO можна виконувати з'єднання з джерелом даних, виконувати команди, отримувати результати. В ADO команди можна виконувати двома різними способами: перший полягає у виконанні методу ExecuteO класу об'єктів Connection, другий - у застосуванні об'єктів

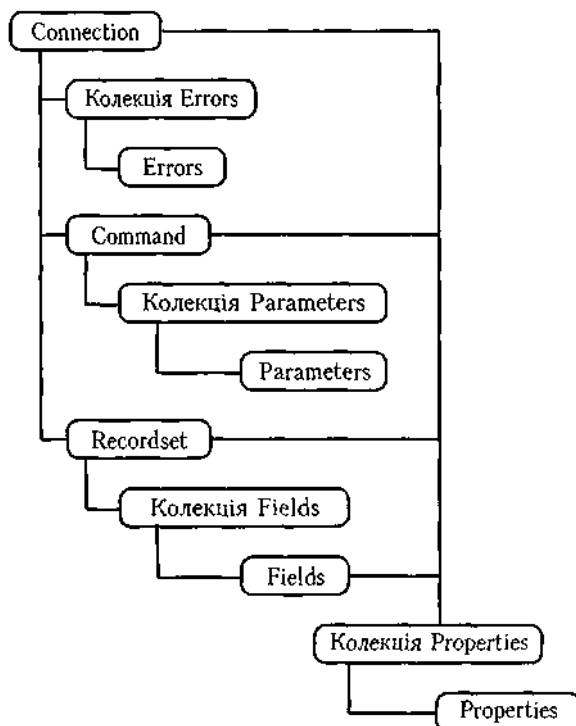


Рис. 3

Command для виконання більш складних команд. Наведемо приклад виконання простої команди з використанням методу Execute().

```

_RecordsetPtr pRS;
variant_t vRowsAffected;
pRs = m_piConnection -> Execute("
INSERT INTO MyTable VALUES (10, 'Ten',
120)",
&vRowsAffected, adCmdText);

```

Висновки щодо застосування OLE DB та ADO. Як показує досвід, додатки OLE DB є досить складними. Цей інтерфейс може виявитися настільки складним для користувачів, що вони не виявлять бажання працювати безпосередньо з інтерфейсом OLE DB. Об'єкти ADO забезпечують більш простий інтерфейс при роботі з джерелом даних OLE DB, тому для більшості ПК легше використати об'єкти даних ActiveX.

Висновки. В останні роки досить успішно розвиваються і впроваджуються технології доступу до БД в об'єктно-орієнтованих мовах програмування. Вибираючи відповідну технологію при розробці ПК, необхідно враховувати тип джерела даних, наявність відповідних засобів для обробки даних, частоту звертання до джерела даних, складність операцій обробки, простоту використання. При програмуванні у середовищі Visual C++ на практиці частіше використовуються такі технології доступу до баз даних, як ODBC та DAO. Проте сучасні технології OLE DB та ADO, які спираються на інтерфейс COM, поступово витісняють DAO. При цьому для більшості ПК використання ADO є простішим порівняно з OLE DB.

1. Эйзенберг Э., Мелтон Дж. Связывания для объектных языков. Открытые системы. - 1999. - № 4.
2. Сторінка драйверів JDBC фірми Oracle - <http://www.oracle.com/st/products/jdbc/sqlj/>.
3. Головна сторінка SQLJ - <http://www.sqlj.org>.

4. Мюллер Дж. Visual C++ 5. Руководство для профессионалов. / Пер. с англ. - СПб.: BHV, 1998. - 720 с, ил.
5. Visual C++5. Руководство разработчика./ Пер. с англ. Д.Бснстидр. - К.; М.; СПб.: Диалектика, 1998. - 768 с, ил.

S.O. Spasiteleva

ACCESS TOOLS TO DATA BASES IN OBJECT-ORIENTED PROGRAMMING LANGUAGES

Article is devoted to problem of compatible using of object oriented programming languages and relational databases. Explored different access technologies to data bases, which use attached to development of program systems by C++ languages and Java. Are Analysed access technologies possibilities to various data sources, their shortages and advantages.