# I. ПРИНЦИПЫ ПРОВЕДЕНИЯ ПРАКТИКИ

## Формулировка задачи и требования

Основными целями прохождения практики в данной компании было усовершенствование знаний в области баз данных, основы которых были получены в университете, углубление знаний СУБД Oracle 11, C++ и применение всех полученных знаний в разработке реального приложения.

Повторение полученных знаний в университете о базах данных и их углубление проводились при чтении необходимой литературы и просмотру видео курсов, предоставленных куратором в компании.

В качестве небольших тестовых задач были даны определенные задачи на усвоение материала по SQL и PL/SQL. К подобным задачам относились задания по созданию и удалению таблиц, добавлению и обновлению данных, написание и работа с запросами SQL.

Примеры задач:

1. Вывести сотрудников с самым высоким и самым низким окладом в разных отделах

Решение:

SELECT DEPT\_NAME,EMPL\_LASTNAME,EMPL\_FIRSTNAME,SALARY

FROM EMPLOYEES,DEPARTMENTS

WHERE SALARY IN(

SELECT MAX(SALARY)

FROM EMPLOYEES

WHERE DEPARTMENTS.DEPT\_ID=EMPLOYEES.DEPT\_ID

GROUP BY DEPT\_NAME) ORDER BY SALARY

SELECT DEPT\_NAME,EMPL\_LASTNAME,EMPL\_FIRSTNAME,SALARY MIN\_SALARY

FROM EMPLOYEES,DEPARTMENTS

WHERE SALARY IN

(SELECT MIN(SALARY) FROM EMPLOYEES WHERE DEPARTMENTS.DEPT\_ID=EMPLOYEES.DEPT\_ID GROUP BY DEPT\_NAME)

ORDER BY SALARY;

1. Увеличить всем сотрудникам оклад на заданную сумму, заданное количество процентов

Решение:

UPDATE EMPLOYEES

SET SALARY=SALARY+1000;

UPDATE EMPLOYEES

SET SALARY=SALARY+SALARY\*0.2;

1. Распределение фонда з/п по отделам в процентах от общего

Решение:

select dept\_name, round(sum(salary)/((select sum(salary) from employees)/100),3)||'%' fond\_of\_department

from departments,employees

where departments.dept\_id=employees.dept\_id

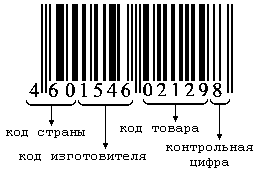
group by dept\_name

Данные решения небольших задач стали результатом полного усвоения материала по SQL и переходом на следующий уровень изучения материала – языка PL/SQL.

Основным проектом по практике было разработка небольшого интерфейса для работы с базой данных и написание пакета на PL/SQL для решения определенной задачи.

Задача состояла в следующем: так как платформа UNA.md в основном используется в сфере торговли, наиболее часто встречающаяся проблема в торговле является появление двух и более товаров с одним и тем же штрих кодом.

Штриховой код или штрих-код - это наносимая на упаковку в виде штрихов закодированная информация, считываемая при помощи специальных устройств. C помощью штрихового кода кодируют информацию о некоторых наиболее существенных параметрах продукции. Наиболее распространены американский Универсальный товарный код UPC и Европейская система кодирования EAN. Согласно той или иной системе, каждому виду изделия присваивается свой номер, состоящий чаще всего из 13 цифр (EAN-13).



Чаще всего проблема дублирования штрих кода в базе данных является существенной проблемой и может нести за собой некоторые последствия для предприятия. В связи с этим возникает необходимость разработки некоторых мер.

# II. ПРОЕКТИРОВАНИЕ И ОПИСАНИЕ РЕАЛИЗАЦИИ ЗАДАЧИ

Первая и самая необходимая мера для устранения данной проблемы является внесение ограничения уникальности для столбца в базе данных, содержащего коды товаров.

Однако если на таблицу не нанесено ограничение уникальности и каким то образом в таблице появляются дубликаты по каким то полям, возникает необходимость разработки некоторого способа вывода и/или удаления дублирующих строк из таблицы.

Для работы с дубликатами была выбрана тестовая база реального торгового предприятия. Из базы данных были выбраны две таблицы, связанные внешним ключом. Первая таблица «vms\_univers» представляет собой справочник товаров, содержащая 7 полей. Вторая таблица «tms\_mpt\_barcode» является справочником штрих кодов и содержит два поля: код самого штрих кода, являющийся первичным ключом и связывающий данную таблицу с таблицей товаров, и поле, содержащее собственно штрих код товара.

*Структура таблиц:*

**Vms\_univers:**

* **Cod** – электронный ключ (первичный ключ)
* **Denumirea –** наименование товара на румынском языке
* **Namerus -** наименование товара на русском языке
* **Um –** единица измерения
* **Codvechi** – артикул
* **Tip**
* **Gr1**

**Tms\_mpt\_bacode:**

* **Cod** – первичный ключ
* **Barcode –** штрих код

Программа была написана с использованием основных понятий PL/SQL.

**Код пакета:**

CREATE OR REPLACE PACKAGE mg\_dupl AS

PROCEDURE dupl\_view;

PROCEDURE dupl\_del;

END mg\_dupl;

CREATE OR REPLACE PACKAGE BODY mg\_dupl AS

PROCEDURE dupl\_view

IS

CURSOR c1 IS SELECT cod, denumirea, namerus, um, codvechi FROM vms\_univers

WHERE tip=’P’ AND gr1=’TVR’ AND cod IN

(SELECT cod FROM tms\_univers\_barcode

GROUP BY cod

HAVING COUNT(\*)>1)

ORDER BY cod;

tab c1%ROWTYPE;

BEGIN

OPEN c1;

LOOP

FETCH c1 INTO tab;

EXIT WHEN c1%NOTFOUND;

DBMS\_OUTPUT.PUT\_LINE(tab.cod || ‘ ’ || tab.denumirea || ‘ ’ || tab.namerus || ‘ ’ || tab.um || ‘ ’ || tab.codvechi);

END LOOP;

CLOSE c1;

END;

PROCEDURE dupl\_del

IS

CURSOR c\_del IS

SELECT cod, denumirea, namerus, um, codvechi FROM vms\_univers

WHERE tip=’P’ AND gr1=’TVR’ AND cod IN

(SELECT cod FROM tms\_mpt\_barcode

GROUP BY cod

HAVING COUNT(\*)>1)

ORDER BY cod;

tab c\_del%ROWTYPE;

BEGIN

OPEN c\_del;

LOOP

FETCH c\_del INTO tab;

EXIT WHEN c\_del%NOTFOUND;

DELETE FROM vms\_univers WHERE vms\_univers.cod = tab.cod;

END LOOP;

CLOSE c\_del;

END;

END mg\_dupl;

Так как данная программа должна быть проверена на реальной базе и таблицах, основным проектом было создание небольшого пользовательского интерфейса для работы с таблицами базы данных.

Существуют два основных стиля интерфейсов пользователя: интерфейс с одним документом (single-document interface, SDI) и интерфейс со многими документами (multiple-document interface, MDI). Текстовый редактор с интерфейсом SDI позволяет открыть только один документ - чтобы открыть другой, следует закрыть предыдущий. В приложении MDI Вы можете открыть сразу несколько документов.

MDI приложение является наиболее подходящем вариантом для работы с базами данных. Приложение было написано на C++ в Embarcadero RAD Studio 10.2 Tokyo.

**Embarcadero RAD Studio** — среда быстрой разработки приложений (RAD) фирмы Embarcadero Technologies, работающая под Windows.Текущая версия *Embarcadero RAD Studio 10.2 Tokyo* объединяет Delphi и C++ Builder в единую интегрированную среду разработки.

Первой формой, запускаемой при открытии приложения, запускается окно авторизации, именуемая далее Registration. Registration – единственная форма приложения, не являющаяся подчиненной формой.

Форма Registration имеет всего два поля для ввода информации: Имя пользователя и Пароль. На данные поля наложено условие входа: поле имени и поле пароля должны иметь значение “twogoose”. При нажатии на кнопку «ОК» происходит проверка условия, и при выполнении данных условий происходит открытие основной формы приложения.

void \_\_fastcall TForm2::Button1Click(TObject \*Sender)

{

AnsiString login=Edit1->Text;

AnsiString password=Edit2->Text;

if(login == "twogoose" && password == "twogoose")

{

ModalResult = mrOk;

}

}

При нажатии на кнопку «Отмена» происходит закрытие всего приложения.

void \_\_fastcall TForm2::Button2Click(TObject \*Sender)

{

Close();

}

Создадим основную MDI форму, которая будет служить контейнером для подчиненных форм. Подчиненная форма - это обычная форма, у которой значение свойства MDIChild равно True. В режиме выполнения подчиненные окна помещаются внутри родительского окна MDI формы.

TForm1 \*Form1;

//---------------------------------------------------------------------------

\_\_fastcall TForm1::TForm1(TComponent\* Owner)

: TForm(Owner)

{

}

//---------------------------------------------------------------------------

Void \_\_fastcall TForm1::Exit1Click(TObject \*Sender)

{

Close();

}

//---------------------------------------------------------------------------

Создадим меню команд MDI. Меню команд MDI формы может быть общим для всех подчиненных окон, или, если подчиненная форма имеет собственное меню команд, оно замещает меню формы контейнера. При закрытии формы-контейнера закрываются все подчиненные формы.

Добавим в заголовочный файл с расширением .h все необходимые элементы меню:

class TForm1 : public TForm

{

\_\_published: // IDE-managed Components

TPanel \*Panel1;

TMainMenu \*MainMenu1;

TMenuItem \*F1;

TMenuItem \*Edit1;

TMenuItem \*Window1;

TMenuItem \*Close1;

TMenuItem \*Saveas1;

TMenuItem \*Exit1;

TMenuItem \*Cut1;

TMenuItem \*Copy1;

TMenuItem \*Paste1;

TMenuItem \*Cascade1;

TMenuItem \*ileHorizontally1;

TMenuItem \*ileVertically1;

TMenuItem \*MinimizeAll1;

TMenuItem \*ArrangeAll1;

……………

};

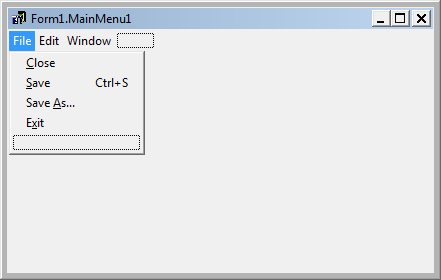


Рис. 1 Конструктор Главного меню 1

Добавим также элемент ToolBar для более удобного манипулирования документами.

TToolBar \*ToolBar2;

TToolButton \*ToolButton2;

TToolButton \*ToolButton3;

TToolButton \*ToolButton4;

TToolButton \*ToolButton5;

TToolButton \*ToolButton6;

TToolButton \*ToolButton7;

TToolButton \*ToolButton8;

TToolButton \*ToolButton10;

TToolButton \*ToolButton11;

Создадим действия для кнопок меню и toolbar-а, добавив в форму три основных элемента: OpenDialog, ActionList, ImageList.

OpenDialog – компонент для открытия файлов.

ActionList – диспетчеризация действий. В компонент **ActionList**,

расположенный в библиотеке на странице Standard, заносится список действий. Перенесем на форму этот компонент и компонент **ImageList**, будем ссылаться на него в свойстве Images компонента **ActionList**. Каждое действие, которое мы вносим в список - это объект типа TAction. Для каждого действия надо задать надпись (Caption), которая далее будет появляться в инициаторах действия кнопках, разделах меню и т.д.

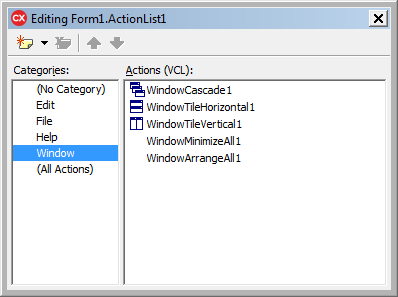


Рис. 2 Конструктор ActionList 1

Создадим панель и добавим на нее два элемента SpeedButton для переключения между двумя подчиненными формами, которые будут находиться внутри родительского окна. Для этого, для начала создадим две Child формы, отображающие необходимые таблицы из базы данных.

Подключение к серверу будем осуществлять через FireDAC. FireDAC – высокопроизводительная библиотека доступа к данным для разработки приложений для нескольких устройств, подключенных к корпоративным базам данных. Наборы данных FireDAC построены на основе мощного механизма доступа к данным. Этот легкий, эффективный и гибкий движок может использоваться непосредственно в приложениях и служит мощным фундаментом API-интерфейсов данных.

Добавим в Data Explorer подключение к серверу, вводя необходимые данные для подключения.

Создадим первую форму, отображающую информацию из двух таблиц, связанных между собой: Справочник товаров и справочник баркодов.

Расположим на форме необходимые компоненты для работы с базой данных и таблицами: DBGrid, FDPhysOracleDriverLink, TestConnection, DataSource, FDQuery.

1)В компоненте FDPhysOracleDriverLink, необходимой для подключения, в свойстве DriverID установим значение Ora, определив подключение к СУБД Oracle.

2)В TestConnection в свойстве ConnectionDefName введем имя нашего подключения. В нашем случае значение подключения равно “test”. В свойстве DriverName так же установим значение нашего СУБД Oracle и в Connected не забудем изменить значение на True.

3)DBGrid будет основным источником отображения информации. При начальном открытии формы грид будет отображать информацию из таблицы Товаров. Для этого добавим первый компонент FDQuery1, установим в свойстве Connection значение компоненты, хранящей подключение к серверу – TestConnection. Зададим в конструкторе необходимый запрос для вывода информации:

select cod,denumirea,namerus,um,codvechi from vms\_univers

where tip='P' and gr1='TVR'

Установим в компоненте свойство Active значение True.

4)Последним шагом для успешного вывода информации в грид свяжем DBGrid с компонентой DataSource1, принимающей и обрабатывающей запросы и данные. В свойство DataSet установим значение компоненты, содержащей наш запрос, в нашем случае FDQuery1.

5) В свойство DataSource компоненты DBGrid установим значение нашего источника данных - DataSource1 и компонента автоматически выгружает источник данных в наш грид.

Таким образом, первая таблица из базы данных выгружена. Теперь добавим в данную форму возможность навигации и переключения между двумя таблицами: vms\_univers и tms\_mpt\_barcode.

1)Добавим группу RadioButton-ов, добавив два элемента: RadioButton1, RadioButton2. Зададим компоненте RadioButton1 название «Справочник товаров», добавив это значение в свойство Caption, а для RadioButton2 – «Справочник баркодов». Таким образом, реализуем для каждой кнопки типа RadioButton определенные события при ее нажатии.

2)Для RadioButton1 компоненты DataSource и FDQuery останутся те же, что были добавлены выше. А для RadioButton2 добавим новые две компоненты DataSource2 и FDQuery2. Запрос для FDQuery2 будет следующим:

select cod,barcode from tms\_mpt\_barcode

3)Напишем код, обрабатывающий событие нажатия на каждую из кнопок:

void \_\_fastcall TForm3::RadioButton2Click(TObject \*Sender)

{

DBGrid1->DataSource = DataSource1;

}

//---------------------------------------------------------------------------

void \_\_fastcall TForm3::RadioButton1Click(TObject \*Sender)

{

DBGrid1->DataSource = DataSource2;

}

Теперь, в зависимости от выбранной кнопки, значение свойства DataSource в компоненте DBGrid1 будет меняться на определенный набор данных.

Для удобства использования таблиц базы данных добавим в наше приложение возможность выбора сортировки таблицы по определённым полям. Так как таблица баркодов состоит всего из двух полей, создадим сортировку только для таблицы товаров, те есть для vms\_univers.

Аналогичным образом добавим на форму две кнопки: «По наименованию» и «По коду». Добавим для каждой кнопки компоненты DataSource и FDQuery.

Запрос для сортировки по наименованию:

void \_\_fastcall TForm3::Button3Click(TObject \*Sender)

{

DBGrid1->DataSource = DataSource5;

}

Запрос для сортировки по коду:

void \_\_fastcall TForm3::Button2Click(TObject \*Sender)

{

DBGrid1->DataSource = DataSource4;

}

Теперь добавим основной функционал данной формы: вывод дубликатов из таблицы vms\_univers по полю cod. Аналогичным образом добавим кнопку и компоненты DataSource и FDQuery и напишем запрос для вывода дубликатов:

select cod,denumirea,namerus,um,codvechi

from vms\_univers

where cod in

(select cod

from tms\_mpt\_barcode

group by cod

having count(\*)>1)

После создания формы товаров создадим подобным образом форму для вывода информации о клиентах. Добавим на форму элементы: DBGrid, DataSource, FDQuery и настроим их, как описано выше.

Запрос на вывод информации о клиентах в грид:

select cod,denumirea,namerus

from vms\_univers

where tip='O' and gr1='E'

Добавим для таблицы сортировку по наименованию:

select cod,denumirea,namerus

from vms\_univers

where tip='O' and gr1='E'

order by denumirea

и по коду:

select cod,denumirea,namerus

from vms\_univers

where tip='O' and gr1='E'

order by cod

Код обработки событий при нажатии на кнопки:

void \_\_fastcall TForm4::Button2Click(TObject \*Sender)

{

DBGrid1->DataSource = DataSource3;

}

//---------------------------------------------------------------------------

void \_\_fastcall TForm4::Button1Click(TObject \*Sender)

{

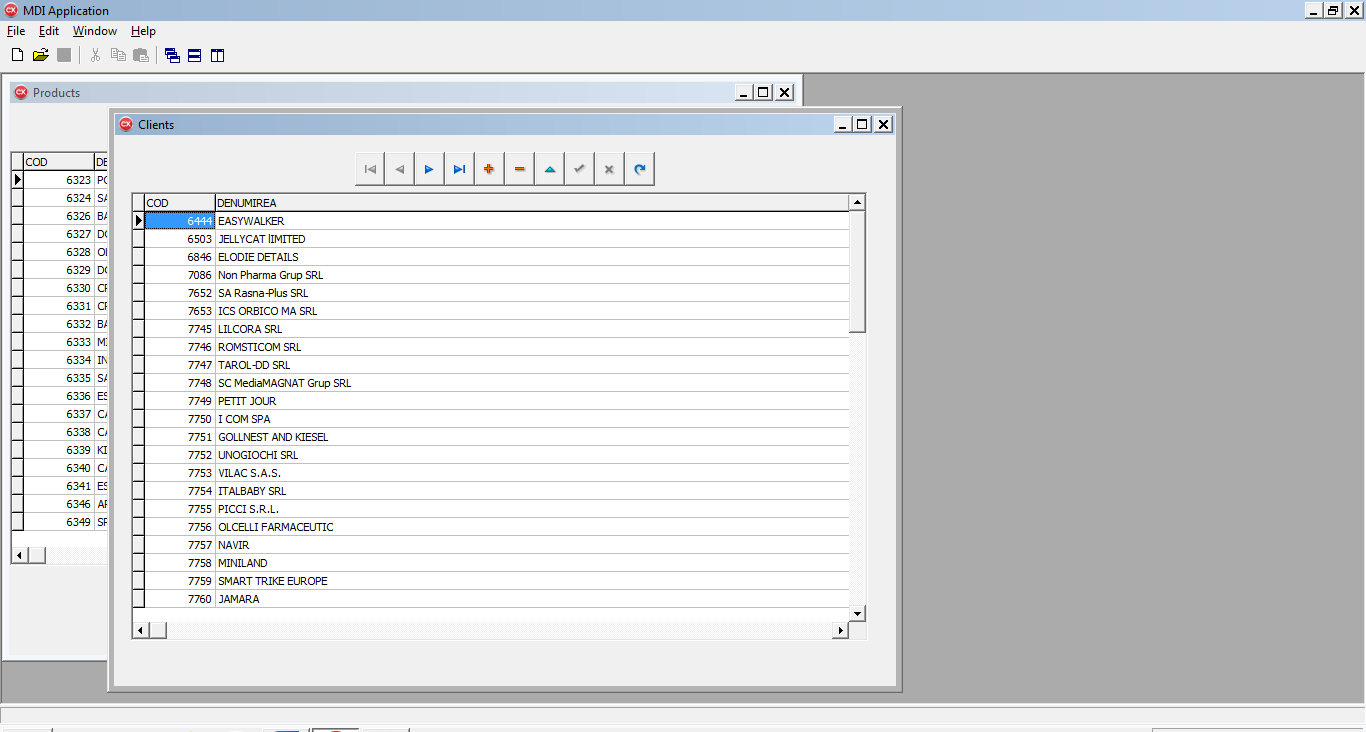
DBGrid1->DataSource = DataSource2;

}

Добавим в данную форму новую компоненту – элемент для навигации по гриду: DBNavigator.

void \_\_fastcall DBNavigator1Click(TObject \*Sender, TNavigateBtn Button);

Компонент DBNavigator обеспечивает перемещение указателя текущей записи, активизацию режима редактирования, добавление и удаление записей. Компонент представляет собой совокупность командных кнопок.



Описание кнопок DBNavigator:

|  |  |  |
| --- | --- | --- |
| Кнопка | Обозначение | Действие |
| http://programm.ws/delphi/Glava2/23.first.gifК первой | nbFirst | Указатель текущей записи перемещается к первой записи файла данных |
| http://programm.ws/delphi/Glava2/23.prior.gifК предыдущей | Prior | Указатель текущей записи перемещается к предыдущей записи файла данных |
| http://programm.ws/delphi/Glava2/23.next.gifК следующей | nbNext | Указатель текущей записи перемещается к следующей записи файла данных |
| http://programm.ws/delphi/Glava2/23.last.gifК последней | nbLast | Указатель текущей записи перемещается к последней записи файла данных |
| http://programm.ws/delphi/Glava2/23.insert.gifДобавить | nblnsert | В файл данных добавляется новая запись |
| http://programm.ws/delphi/Glava2/23.delete.gifУдалить | nbDelete' | Удаляется текущая запись файла данных |
| http://programm.ws/delphi/Glava2/23.edit.gifРедактирование | nbEdit | Устанавливает режим редактирования текущей записи |
| http://programm.ws/delphi/Glava2/23.cancel.gifСохранить | nbPost | Изменения, внесенные в текущую запись, записываются в файл данных |
| http://programm.ws/delphi/Glava2/23.cancel.gif Отменить | Cancel | Отменяет внесенные в текущую запись изменения |
| http://programm.ws/delphi/Glava2/23.refresh.gifОбновить | nbRefresh | Записывает внесенные изменения в файл |

# III. ТЕСТИРОВАНИЕ И ПРОВЕРКА ТРЕБОВАНИЙ

Приложение было протестировано на реальной базе и с использованием реальных таблиц и данных. Данное разработанное приложение является необходимым минимумом для работы с данными. В последующем приложение можно развивать для дальнейшего использования, расширение возможностей для пользования с большим количеством данных.

При разработке приложения были приняты во внимание все необходимые требования.

# IV. ВЫВОДЫ И РЕЗУЛЬТАТЫ

В результате прохождения практики были выполнены все основные цели и задачи, поставленные программой практики, теоретические знания, полученные во время учебы, подкреплены практическими. Были получены навыки в разработке пользовательских приложений, изучены методы разработки, используемые в реальной сфере программирования. Были получены базовые знания, необходимые для работы в новой для меня среде разработки Embarcadero RAD Studio.

В ходе курсового проектирования было самостоятельно разработано:

техническое задание на программное средство;

программное средство в соответствии с поставленной задачей.

Сфера, в которой проводилась практика, меня довольно заинтересовала и я считаю необходимым и дальше расширять свои знания в области разработки баз данных и разработке комплексных информационных систем.

# V. ПРИЛОЖЕНИЯ

