МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ
РОССИЙСКОЙ ФЕДЕРАЦИИ

ФЕДЕРАЛЬНОЕ АГЕНТСТВО ПО ОБРАЗОВАНИЮ

КУРГАНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

Кафедра информационных технологий
и методики преподавания информатики

Базы данных
«КЛИЕНТ – СЕРВЕР»

Методические рекомендации

для студентов специальностей 010101, 050202

(часть II)

Курган 2009

Кафедра: «Информационные технологии и методика преподавания информатики»

Дисциплина: «Базы данных «Клиент-сервер»
(специальности 010101, 050202)

Составитель: старший преподаватель С.Г. Тетюшева

Утверждены на заседании кафедры «*01*» *октября* 200*8* г.

Рекомендованы методическим советом университета
«*10*» *июня* 200*9* г.

ЛАБОРАТОРНАЯ РАБОТА №4
УТИЛИТА DATABASE EXPLORER

Вызов программы SQL Explorer осуществляется из главного меню Windows командой ***Программы\Borland Delphi\SQL Explorer***. Окно программы представлено на рис. 1. В левой панели на странице Databases вы можете видеть дерево зарегистрированных псевдонимов. Выделив интересующий вас псевдоним (на рис. 1 это псевдоним *newsklad*), вы увидите на правой панели его параметры. При желании вы можете их изменить.

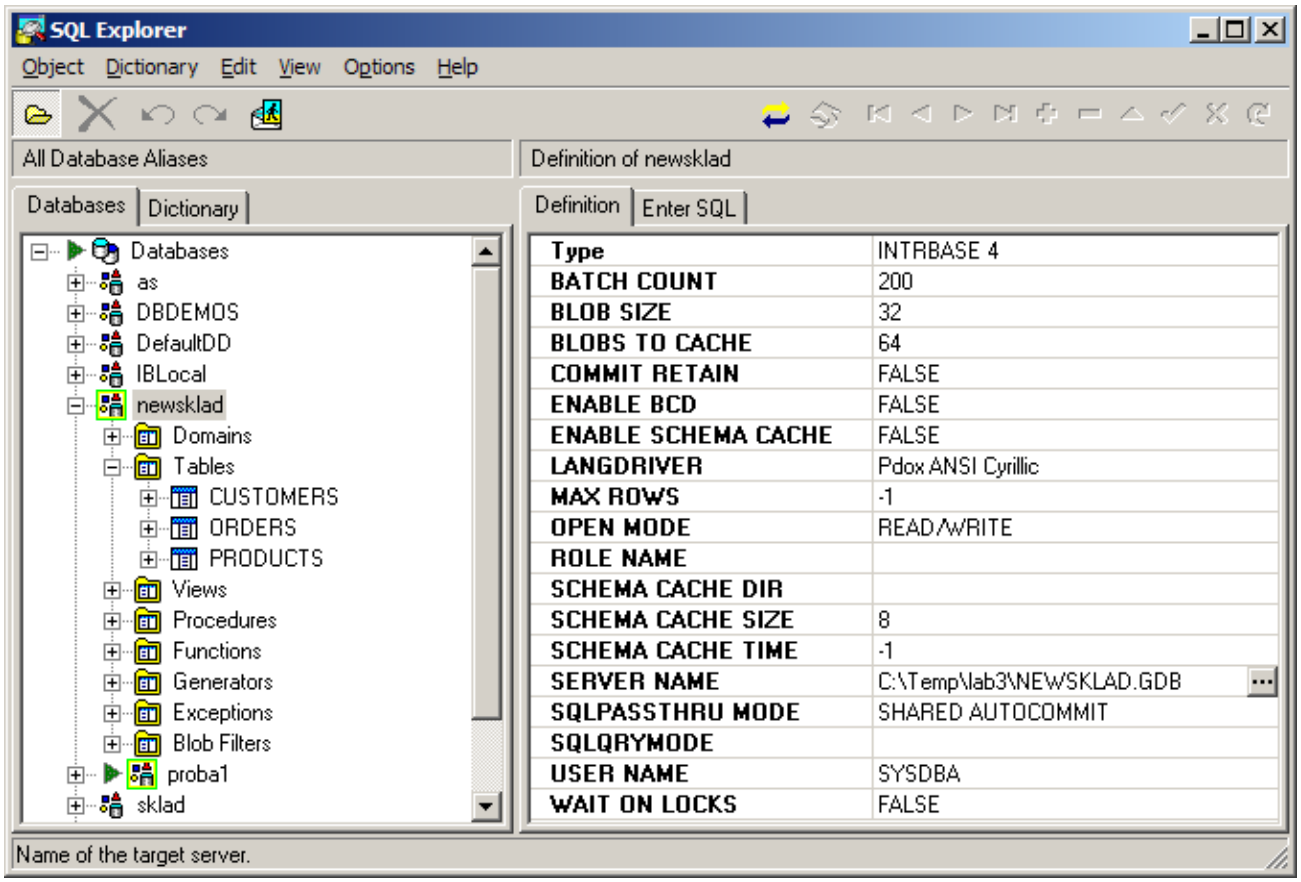

Рис. 1. Окно SQL Explorer – просмотр параметров псевдонима

Для создания нового псевдонима БД нужно:

1. Выделить корневой узел *Databases* в левой панели.

2. В меню утилиты выберите команду ***Object\New***.

3. Перед вами появится небольшое диалоговое окно, показанное на рис. 2. В его выпадающем списке вы должны выбрать для создаваемого псевдонима драйвер **INTRBASE**.

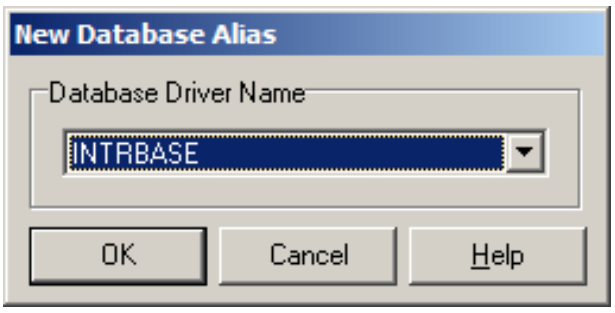

Рис. 2. Диалоговое окно выбора драйвера для нового псевдонима

4. Выбрав драйвер, щелкните на **ОК**, и в дереве псевдонимов появится новая вершина, для которой нужно задать имя — псевдоним.

5. В правой панели утилиты появятся параметры, которые вы должны установить для создаваемого псевдонима. Для типа INTRBASE укажите следующие параметры:

SERVER NAME	Путь к базе данных
USER NAME	Имя владельца БД (для администратора – SYSDBA)

6. После задания параметров щелкните правой кнопкой мыши по новому псевдониму и из всплывшего меню выберите команду ***Apply***, подтвердив изменения. Новый псевдоним будет зафиксирован.

Для *удаления* существующего псевдонима выделите его в левой панели, щелкните правой кнопкой мыши и из всплывшего меню выберите команду ***Delete***.

Возможности SQL Explorer по просмотру и модификации баз данных не ограничиваются модификацией существующих и созданием новых псевдонимов.

Для просмотра структуры и содержимого БД в левой панели утилиты SQL Explorer щелкните по кнопке + слева от интересующего вас псевдонима БД, а затем выберите нужный объект, например, таблицы (Tables). После выбора одной из таблиц БД в правой панели на странице ***Definition*** вы можете просмотреть общую информацию о таблице, на странице ***Text*** вы можете увидеть текст запроса SQL, создавшего данную таблицу, на странице ***Data*** вы можете просмотреть хранящуюся в таблице информацию (этот момент отображен на рис. 3). При этом кнопки навигатора вверху диалогового окна позволяют вам редактировать записи, вставлять новые записи и т.п. На странице ***Enter SQL*** можете сформировать и выполнить запрос к таблице БД.

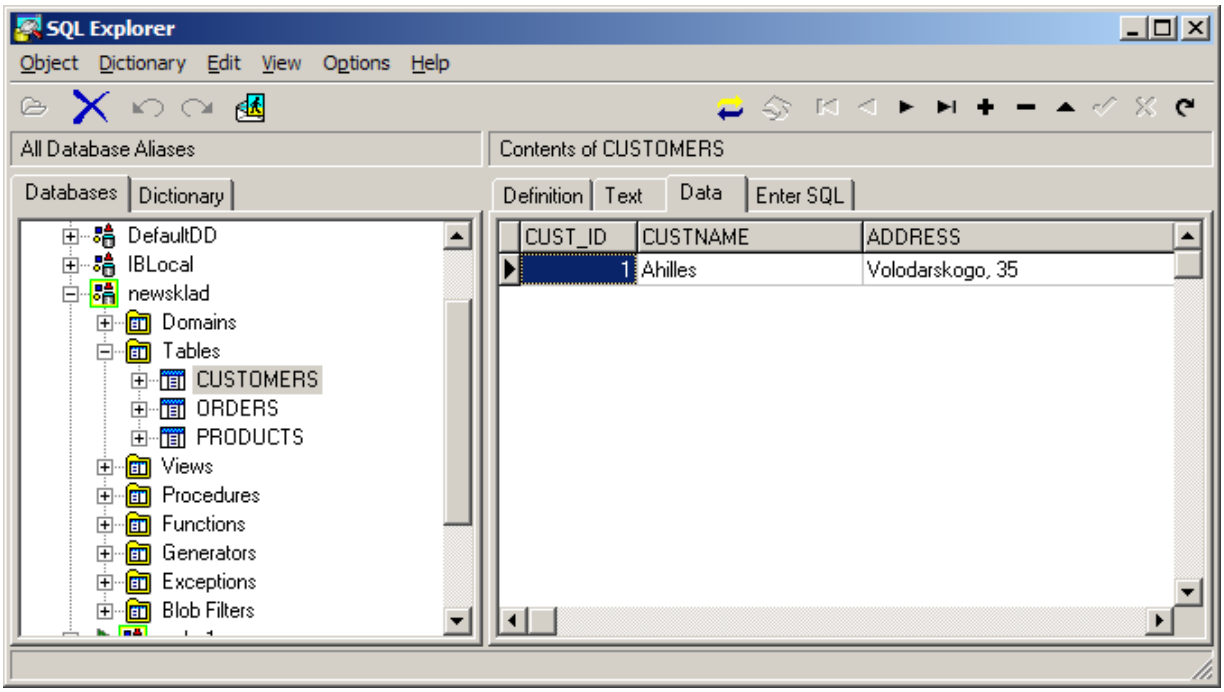

Рис. 3. Окно SQL Explorer – просмотр данных в таблице Customers

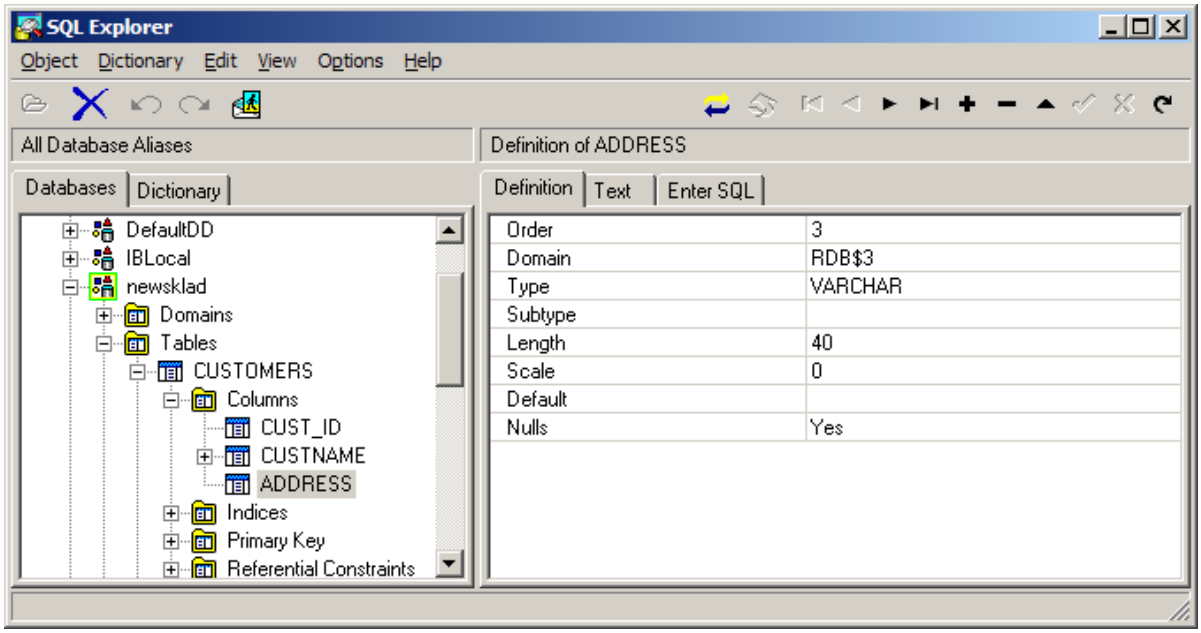

Рис. 4. Окно SQL Explorer – просмотр объявления поля Address

Вы можете продвинуться далее и раскрыть структуру таблицы. На рис. 4 изображен просмотр поля *ADDRESS* таблицы *CUSTOMERS*, причем вы можете изменить соответствующую информацию о нем.

С помощью команды ***Object\New*** вы можете в зависимости от того, на каком уровне просмотра дерева информации вы находитесь, создать новый псевдоним, или новую таблицу, или новое поле существующей таблицы.

Рассмотрим основные операции по созданию и сопровождению таблиц на примере базы данных, содержащей две таблицы *Pers* (сведения о сотрудниках некоторой организации) и *Dep* (сведения о подразделениях некоторой организации) (табл. 2 и табл. 3).

Таблица 2

Пример таблицы Pers

Номер	**Отдел**	**Фамилия**	**Имя**	**Отчество**	**Год рождения**	**Пол**
Num	***Dep***	***Fam***	***Nam***	***Par***	***Yaer_b***	***Sex***
integer	*varchar (20)*	*varchar (30)*	*varchar (20)*	*varchar (30)*	*integer*	*varchar (1)*
1	Бухгалтерия	Иванов	Иван	Иванович	1950	м
2	Цех 1	Петров	Петр	Петрович	1960	м
3	Цех 2	Сидоров	Сидор	Сидорович	1955	м
4	Цех 1	Иванова	Ирина	Ивановна	1961	ж
. . .	. . .	. . .	. . .	. . .	. . .	. . .

Таблица 3

Пример таблицы Dep

Отдел	**Тип**
Dep	***Proisv***
varchar (20)	*varchar (20)*
Бухгалтерия	управление
Цех 1	производство
Цех 2	производство

1. Задайте каталогу, в который вы поместите файл БД, имя **dbP**.

2. Создайте в каталоге **dbP** для пустой пока БД файл **IB_PROBA.GDB** с помощью утилиты **IBConsole**:

- Запустите утилиту (Пуск\Программы\InterBase\IBConsole).
- Подключитесь к локальному серверу, выбрав команду ***Server\Login*** и введя пароль администратора.
- Выполните команду ***Database\Create Database…*** и укажите в открывшемся окне параметры новой БД (рис. 5).
- Закройте окно IBConsole.

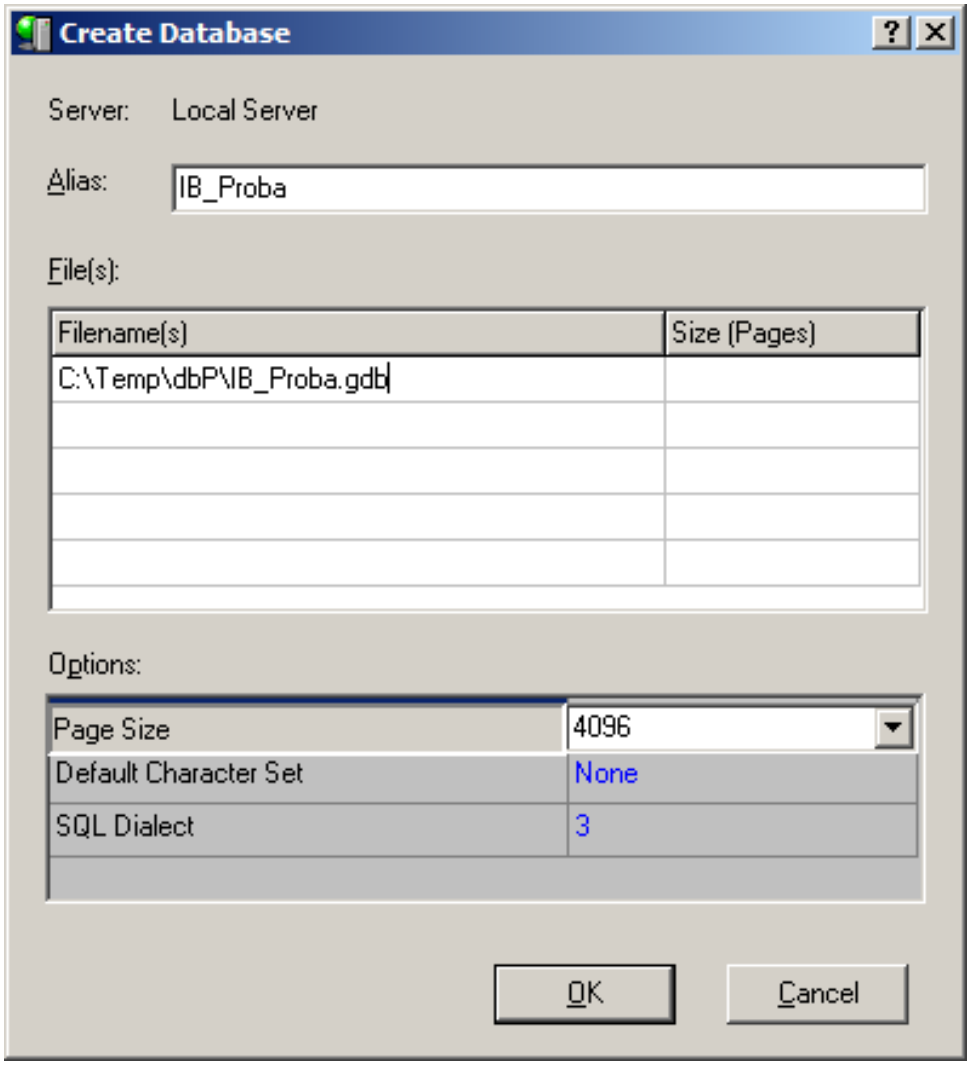

Рис. 5. Создание новой базы данных с помощью утилиты IBConsole

3. Запустите утилиту SQL Explorer (***Программы\Borland Delphi\SQL Explorer***).

4. Создайте псевдоним для новой БД как это описано выше (рис. 6).

5. Создадим таблицу **Pers**, для этого:

- Раскройте структуру БД и выделите объект ***Tables***.
- С помощью команды ***Object\New*** создайте новую таблицу и назовите ее ***Pers***.
- Для создания первого поля раскройте структуру таблицы *Pers*, выделите объект ***Columns***, выполните команду ***Object\New*** и дайте новому объекту

имя ***Num***. В правой панели окна утилиты задайте для поля параметры (рис. 7).

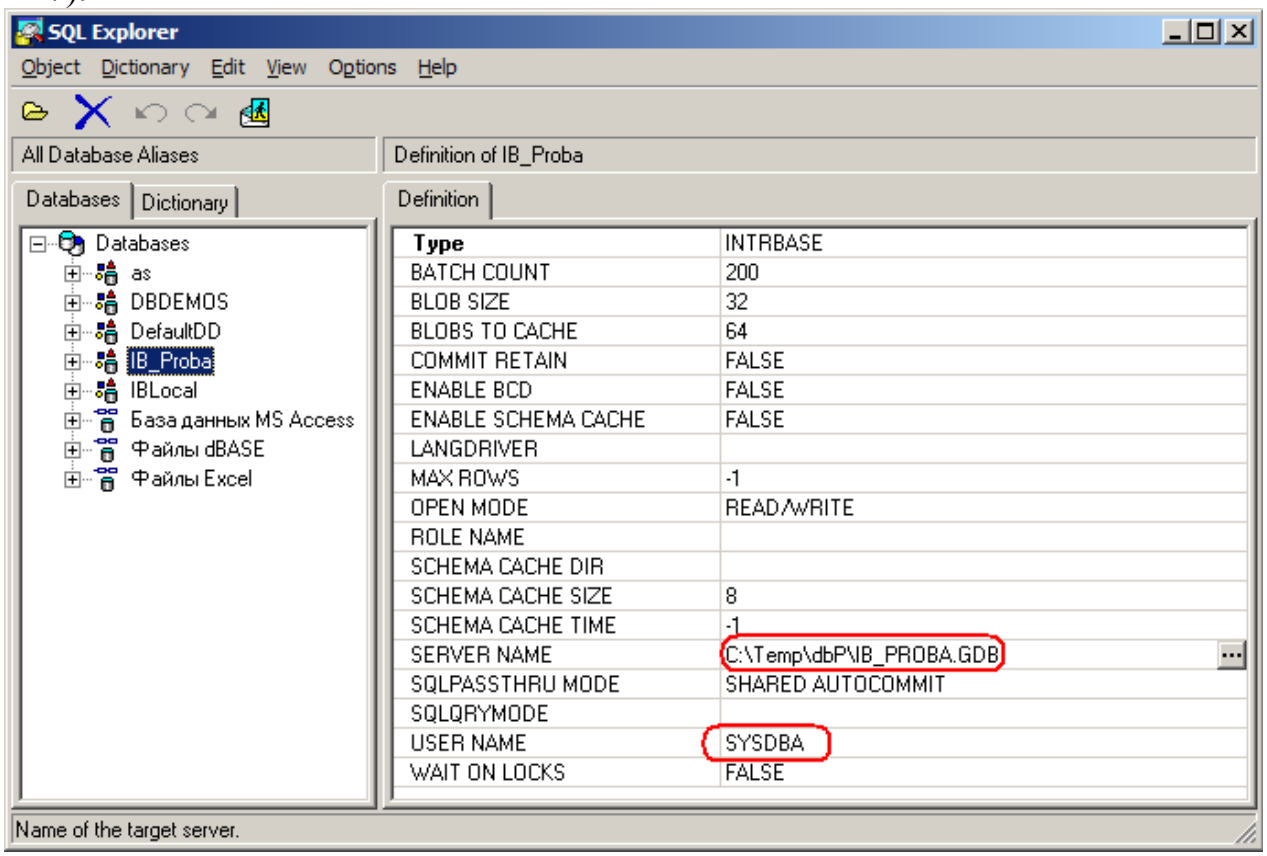

Рис. 6. Создание псевдонима для новой базы данных с помощью утилиты SQL Explorer

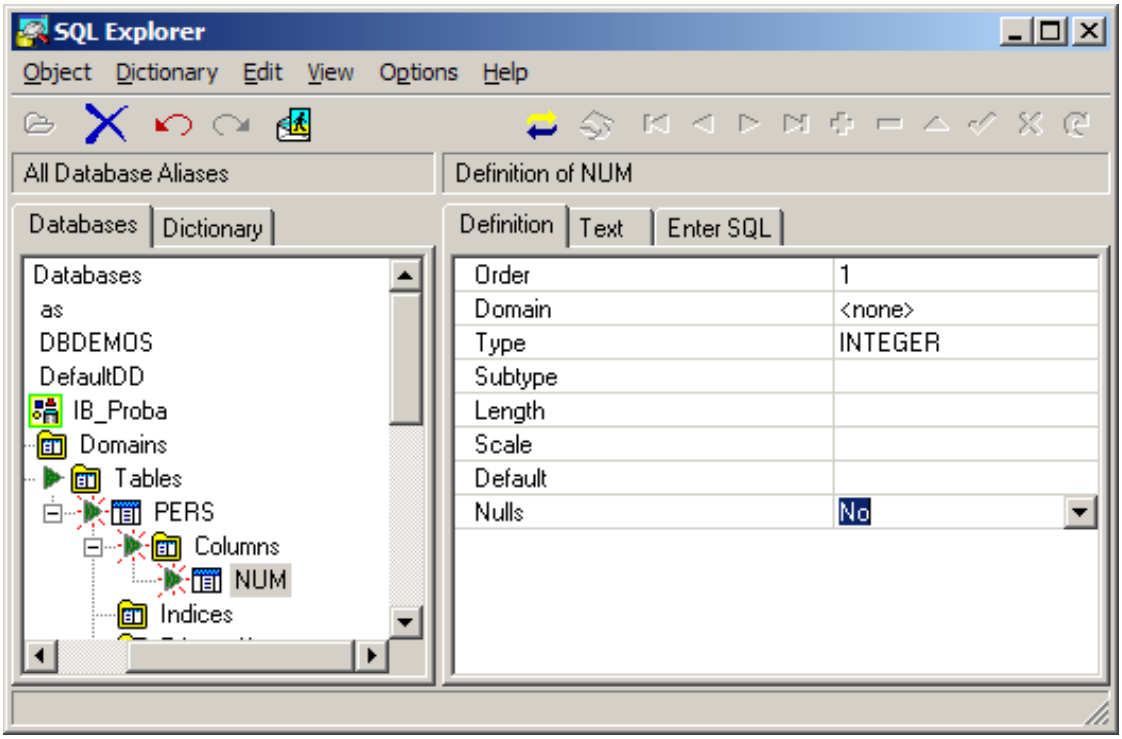

Рис. 7. Создание нового поля таблицы базы данных

- Аналогичным образом добавьте в таблицу ***Pers*** остальные поля согласно табл.1 (рис. 8).

- После создания всех полей щелкните правой кнопкой мыши по новой таблице и из всплывшего меню выберите команду ***Apply***, подтвердив изменения.

6. Аналогичным образом создайте таблицу ***Dep*** (рис. 9).

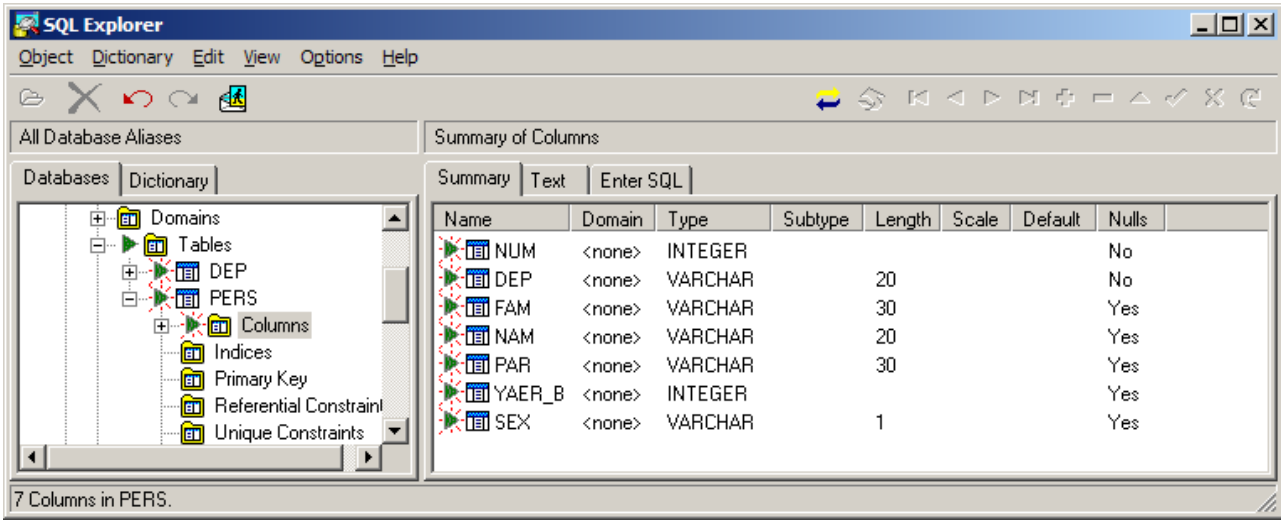

Рис. 8. Структура таблицы Pers

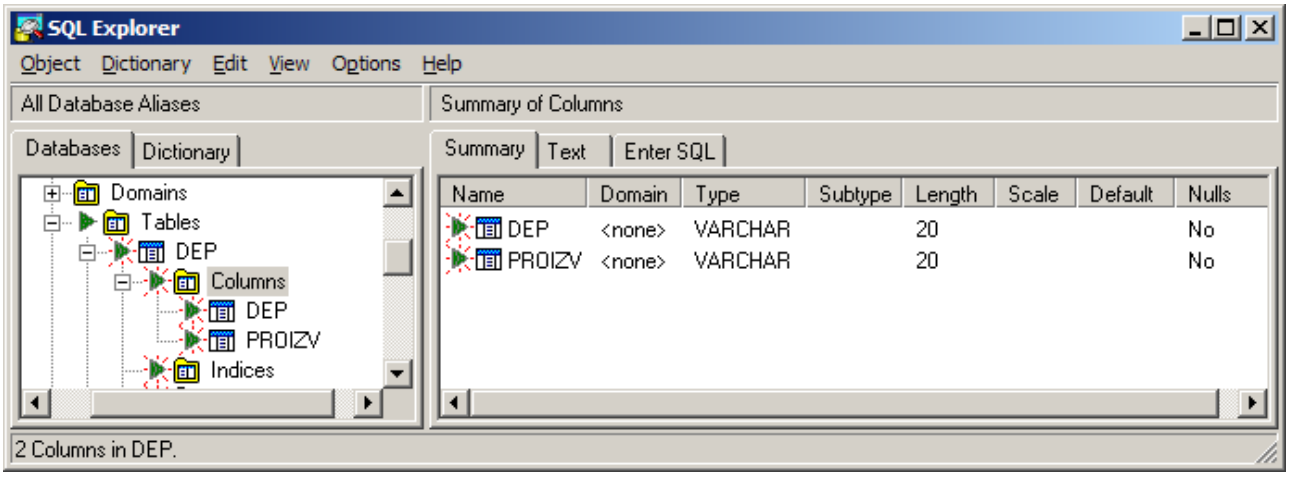

Рис. 9. Структура таблицы Dep

7. Создадим генератор для заполнения поля NUM, являющегося первичным ключом таблицы Pers:

- В структуре БД щелкните правой кнопкой мыши по объекту ***Generators***, выберите команду ***New***, генератору дайте имя **GEN_PERS_ID**. В правой панели на странице *Text* в автоматически созданном SQL-коде *исправьте первоначальное значение для генератора с **0** на **1***.
- Щелкните правой кнопкой мыши по созданному генератору и из всплывшего меню выберите команду ***Apply***, подтвердив изменения.

8. Для вызова генератора создайте хранимую процедуру:

- В структуре БД щелкните правой кнопкой мыши по объекту ***Procedures***, выберите команду ***New***, процедуре дайте имя **SP_GEN_PERS_ID**.
- В правой панели на странице *Text* дополните SQL-код:

```
CREATE PROCEDURE SP_GEN_PERS_ID
  RETURNS (ID  INTEGER)
```

```
AS
BEGIN
  ID = GEN_ID (GEN_PERS_ID, 1);
END
```

- Щелкните правой кнопкой мыши по созданной процедуре и из всплывшего меню выберите команду ***Apply***, подтвердив изменения.

9. Создайте триггер:

- В структуре таблицы ***Pers*** щелкните правой кнопкой мыши по объекту ***Triggers***, выберите команду ***New***, триггеру дайте имя **PERS_BI**.
- В правой панели на странице *Definition* выберите из списка тип триггера **BEFORE INSERT**.
- Затем на странице *Text* дополните SQL-код:

```
CREATE TRIGGER PERS_BI FOR PERS
BEFORE INSERT POSITION 0
AS BEGIN
IF (NEW.Num IS NULL) THEN
    NEW.Num=GEN_ID(GEN_PERS_ID,1);
END
```

- Щелкните правой кнопкой мыши по созданному триггеру и из всплывшего меню выберите команду ***Apply***, подтвердив изменения.

12. Для всех созданных таблиц создайте триггеры для каскадных воздействий (см. лабораторную работу 3).

13. Создайте приложение для просмотра и редактирования содержимого таблиц созданной базы данных.

ЗАДАНИЕ. С помощью утилиты SQL Explorer создайте и затем реализуйте в клиентском приложении хранимую процедуру, позволяющую вывести фамилии и имена сотрудников, не достигших пенсионного возраста (для женщин – 50 лет, для мужчин – 60), сгруппировав данные по подразделениям.

ЛАБОРАТОРНАЯ РАБОТА №5
ЗАЩИТА ТАБЛИЦ СЕРВЕРА INTERBASE

1. Управление пользователями

Каждый сервер InterBase имеет пользователя SYSDBA с административными полномочиями. Сначала это единственный авторизованный пользователь на сервере и он должен авторизовать (зарегистрировать) других пользователей. Только пользователь SYSDBA имеет права на добавление, модификацию и удаление пользователей.

Имена пользователей могут быть длиной до 31 символа, не являются чувствительными к смене регистра, но не должны содержать пробелов. *Пароли чувствительны к регистру и не должны содержать пробелов.* Значение имеют только первые 8 символов пароля, но длина пароля может достигать 32 символов.

С помощью SQL-команды создать или удалить пользователя IntrBase нельзя – это следствие внесения системы безопасности на уровень сервера.

Добавить нового пользователя можно так:

1. В окне утилиты IBConsole выберите команду ***Server\User Security…***

2. В появившемся окне нажмите кнопку **New**.

3. Введите в поля окна диалога имя пользователя (*User Name*), пароль (*Password*), его подтверждение (*Confirm Password*) — т.е. повторите пароль еще раз. В нижних полях редактирования можете (но не обязательно) написать свои имя (*First Name*), отчество (*Middle Name*) и фамилию (*Last Name*) (рис. 10).

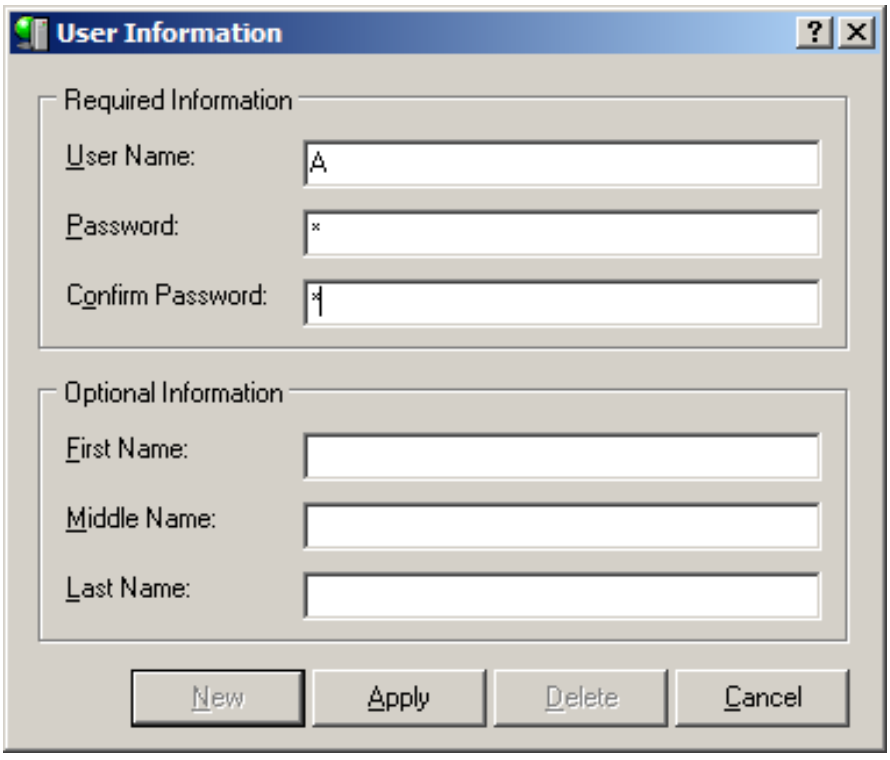

Рис 10. Создание нового пользователя БД

4. Щелкните на **Apply**, а затем кнопку **Close**. В окне утилиты IBConsole в левой области щелкните по элементу *Users*, в правой области окна (рис. 11) должно появиться ваше имя как пользователя.

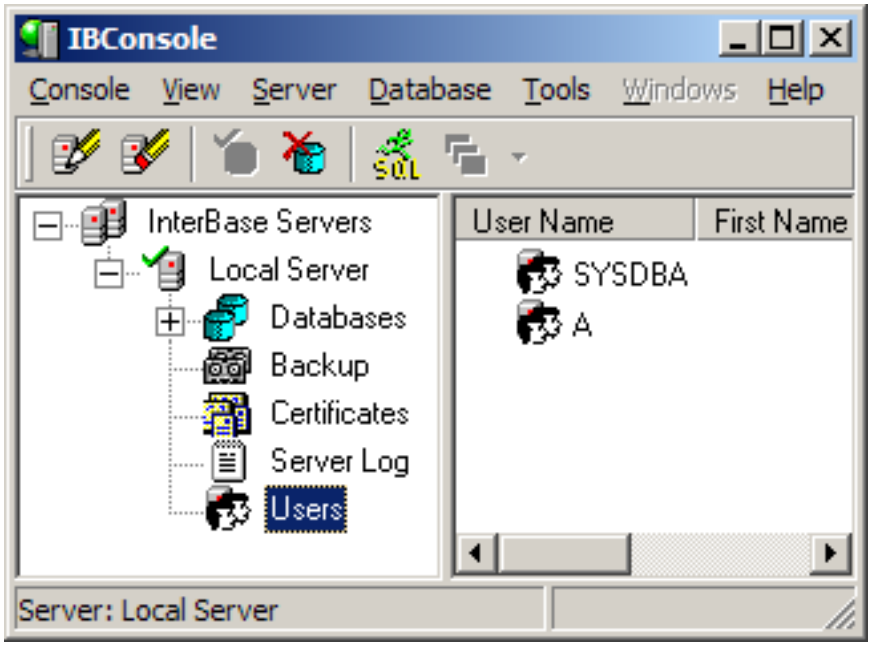

Рис.11. Список пользователей сервера InterBase

В последующем вы сможете изменять пароли пользователей с помощью команды ***Server\| User Sequrity***. В открывающемся диалоговом окне вы можете выбрать из выпадающего списка того пользователя, для которого хотите сменить пароль, и ввести новый пароль. Вообще учтите, что все, что вы сейчас проделывали, это функции системного администратора.

Новый пользователь может создавать базы данных и регистрировать их, после чего он станет их владельцем (*owner*) с правом полного доступа. При соединении с базой данных пользователь будет указывать свои учетные данные. Кроме того, владелец базы данных может предоставлять привилегии доступа к своей БД другим пользователям, разрешая или запрещая им отдельные операции.

Задание для самостоятельной работы

1. Создайте БД «Студенты факультета», состоящую из таблиц «Группы» и «Студенты»:

Таблица «GRUPPA»

Имя поля	*Тип данных*
Gr_Num	integer, not null
FIO_Star	varchar(30)
Kol_Stud	integer

Таблица «STUDENTS»

Имя поля	*Тип данных*
Num_Zach	integer, not null
FIO_Stud	varchar(30)
Gr_Num	integer, not null

2. Создайте приложение для работы с БД «Студенты факультета» (см. рис 12).

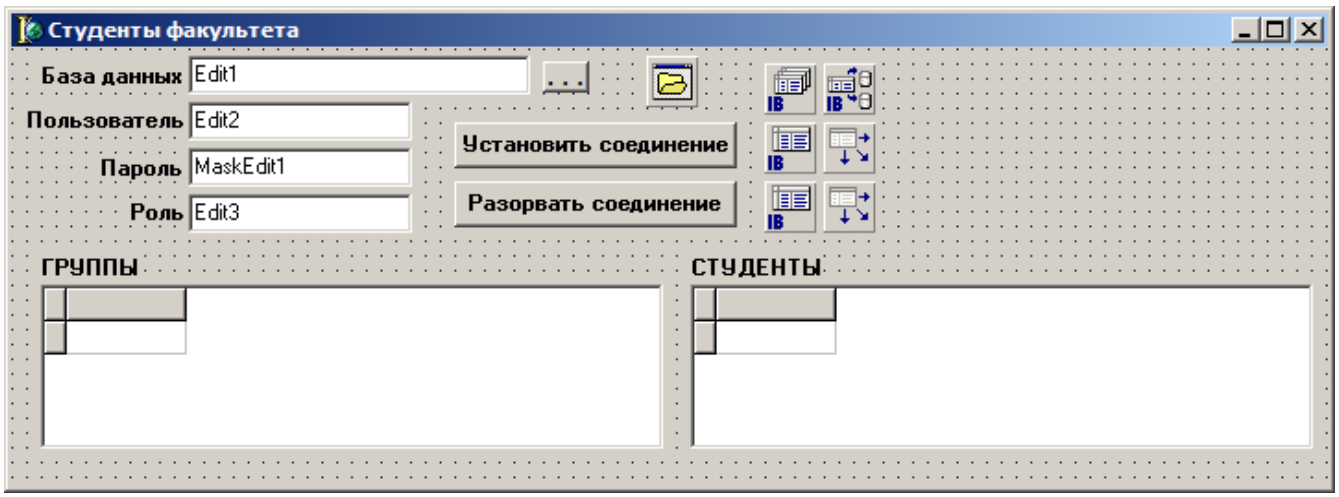

Рис 12. Главная форма приложения

3. В обработчик нажатия кнопки [...] впишите следующий код:

```
if OpenDialog1.Execute then Edit1.Text := OpenDialog1.FileName;
```

4. В обработчик кнопки «Установить соединение» впишите следующий код:

```
IBTransaction1.Active := false;
IBDatabase1.Connected := false;
IBDatabase1.DatabaseName := Edit1.Text;
IBDatabase1.Params.Clear;
IBDatabase1.Params.Add (Format ('USER_NAME=%s', [Edit2.Text]));
IBDatabase1.Params.Add (Format ('PASSWORD=%s', [MaskE-
    dit1.Text]));
if Edit3.Text <> '' then IBDatabase1.Params.Add (Format
    ('SQL_ROLE_NAME=%s', [Edit3.Text] ));
IBDatabase1.Connected := true;
IBTransaction1.Active := true;
IBTable1.Open;
IBTable2.Open;
```

5. В обработчик нажатия кнопки «Разорвать соединение» впишите следующий код:

```
IBTransaction1.Active := false;
IBDatabase1.Connected := false;
```

6. Проверьте работоспособность приложения для пользователя SYSDBA.

7. Зарегистрируйте на сервере InterBase пользователей **A**, **B** и **C**.

2. Предоставление привилегий

Сервер InterBase способен защищать таблицы БД от несанкционированного доступа на основе *прав* и *ролей*. Сразу после создания таблицы правом неограниченного ее использования облает ее собственник (т.е. создавший ее пользователь) и системный администратор SYSDBA. Любой из них может передать некоторые (или все) права владения таблицей другому пользователю или группе пользователей.

Для управления доступом пользователей к базе данных в языке SQL существуют два оператора: GRANT и REVOKE. Как правило, эти операторы используются администратором базы данных или его помощником по безопасности.

2.1. Оператор GRANT

SQL-оператор GRANT назначает привилегии доступа различным субъектам безопасности – конкретным пользователям, ролям, а также хранимым процедурам и триггерам. Объектами безопасности (охраняемыми объектами), к которым применяется GRANT, могут быть таблицы целиком, столбцы таблиц, триггеры, хранимые процедуры, просмотры.

Синтаксис данного оператора имеет следующий вид:

GRANT <привилегия_1> [, <привилегия_2>]
ON <имя_объекта>
TO <имя_пользователя> [WITH GRANT OPTION];

Таблица 4

Привилегии доступа к таблицам InterBase

Привилегия	Разрешает пользователям…
SELECT	Просматривать строки в таблице
DELETE	Удалять строки
INSERT	Вставлять строки
UPDATE	Изменять все или указанные столбцы
EXECUTE	Выполнять хранимые процедуры
REFERENCES	Создавать внешний ключ, который ссылается на указанный первичный ключ таблицы, даже если пользователь не является ее владельцем
ALL	Объединяет все права, кроме EXECUTE

Предоставление пользователю с именем USER права на выбор данных из таблицы СОТРУДНИКИ выполняется с помощью следующего оператора:

GRANT select ON Сотрудники TO user;

С помощью одного оператора GRANT можно задавать сразу несколько привилегий. Например, следующий оператор предоставит пользователю USER право как просматривать, так и добавлять данные в таблицу СОТРУДНИКИ:

GRANT select, insert ON Сотрудники TO user;

Если права передаются любому пользователю, указывают зарезервированное слово PUBLIC:

GRANT all ON books TO PUBLIC;

Если за именем таблицы в круглых скобках указан перечень столбцов, права распространяются только на перечисленные столбцы:

GRANT updates ON Firms (FAdres, FPhone, FEMail) TO Petrov;

При вызове оператора GRANT может использоваться необязательное предложение WITH GRANT OPTION. Данное предложение означает, что поль-

зователь, для которого предоставляются привилегии, также получает право предоставлять привилегии на данный объект. Например, если вызвать рассмотренный выше оператор с предложением WITH GRANT OPTION, то пользователь с именем USER, кроме права просматривать и добавлять данные в таблицу СОТРУДНИКИ, получит также право предоставлять эти привилегии другим пользователям:

GRANT select, insert ON Сотрудники TO user WITH GRANT OPTION;

Аналогичный синтаксис раздачи прав применяется и для хранимых процедур, только в первой части команды GRANT пишется

GRANT EXECUTE ON PROCEDURE <имя_процедуры>

а далее как обычно. Например, для выдачи пользователю TESTUSER разрешения запускать хранимую процедуру SP_ADD надо написать следующее:

GRANT EXECUTE ON PROCEDURE sp_add TO testuser;

Для того чтобы некая процедура SP_MAIN могла вызывать процедуру SP_ADD без наличия таких прав у пользователя, вызывающего SP_ADD, надо написать так:

GRANT EXECUTE ON PROCEDURE sp_add TO sp_main;

Однако раздача прав объектов на использование других объектов нужна только в том случае, если база данных проектируется с намерением скрыть исходные таблицы от пользователей. В противном случае гораздо проще будет раздавать права напрямую пользователям (ролям).

Задание для самостоятельной работы

1. Предоставьте привилегии:

- пользователю ***A*** - только просматривать таблицы БД «Студенты факультета»;
- пользователю ***B*** - просматривать таблицы и редактировать содержимое их столбцов;
- пользователю ***C*** - просматривать таблицы БД а также добавлять и удалять строки в таблицы.

2. Проверьте работу приложения при подключении к БД каждого из зарегистрированных пользователей. При необходимости внесите изменения в проект для обеспечения его работоспособности. Например, для пользователя ***A*** таблицы должны открываться только в режиме просмотра.

2.2. Оператор REVOKE

Оператор REVOKE используется для отмены предоставленных пользователю привилегий. Данный оператор может вызываться с одним из двух параметров — RESTRICT или CASCADE. При использовании варианта RESTRICT оператор REVOKE будет успешно выполнен только в том случае, если его выполнение не приведет к появлению так называемых *оставленных* привилегий.

ПРИМЕЧАНИЕ. *Оставленными называются привилегии, оставшиеся у пользователя, которому они были предоставлены с помощью предложения WITH GRANT OPTION оператора GRANT.*

При использовании режима CASCADE удаляются все привилегии, которые могли бы остаться у других пользователей. Это означает, что если пользователю USER1 были предоставлены привилегии с помощью параметра WITH GRANT OPTION, а он, в свою очередь, предоставил эти привилегии пользователю USER2, то отмена привилегий пользователю USER1 в режиме CASCADE приведет к отмене привилегий и для пользователя USER2.

Синтаксис оператора REVOKE имеет следующий вид:

REVOKE <привилегия_1> [, <привилегия_2>]
ON <имя_обьекта>
FROM <имя_пользователя> [RESTRICT | CASCADE];

Например, для отмены права добавления данных в таблицу СОТРУДНИКИ для пользователя USER следует использовать следующий оператор:

REVOKE insert ON Сотрудники FROM user;

Задание для самостоятельной работы

Пользователю **A** отмените привилегию просматривать таблицы БД «Студенты факультета». Внесите изменения в проект (вывод для пользователя **A** сообщения о недоступности БД) и проверьте работу приложения при подключении к БД пользователя **A**.

3. Использование ролей

Роль – это аналог группы пользователей в Windows 2000/XP. Рекомендуется назначать права именно ролям, а не пользователям. Действительно, роли хранятся в самой базе данных, а пользователи – в служебной базе, которая у каждого сервера своя. Поэтому при переносе БД на другой сервер роли сохраняются, а пользователи – нет. Если права даны не ролям, а пользователям, то потребуется их назначать заново. При использовании же ролей достаточно будет включить в их состав новых пользователей, права менять нет необходимости.

3.1. Создание роли

Для того, чтобы создать роль, необходимо воспользоваться оператором

CREATE ROLE <имя_роли>;

По умолчанию, после создания роль не обладает какими-либо привилегиями. Соответственно, все необходимые права по доступу к объектам базы данных должны быть назначены до использования роли. Учитывая, что создания роли как таковой не прибавляет каких-либо прав, любой пользователь может создать роль. Также, исходя из того, что роль определяется и храниться непосредственно в базе данных, необходимо подключиться к этой базе до создания роли. Пример создания роли:

CREATE ROLE customers;

3.2. Предоставление привилегий роли

Привилегии для роли назначаются с использованием оператора выдачи разрешений, общий синтаксис которого в данном случае имеет следующий вид:

GRANT <привилегия>
ON [TABLE] {<имя_таблицы> | <имя_просмотра>}
TO <имя_роли>;

Для того, чтобы иметь возможность назначить привилегию роли, необходимо быть:

- пользователем SYSDBA;
- владельцем таблицы или представления;
- пользователем, имеющим право назначать привилегии для этой таблицы или представления (т.е. иметь GRANT OPTION).

Примеры предоставления объектных прав роли:

GRANT all ON test_scores TO full_access;
GRANT insert, select ON TABLE employee TO bjones;

3.3. Назначение роли пользователю

После того, как роль была создана и ей были выданы необходимые привилегии, нужно назначить роль пользователю. Когда пользователь при подключении к базе данных указывает роль, он приобретает все привилегии, предоставленные этой роли. Итоговые привилегии получаются суммированием привилегий, непосредственно выданных пользователю, и привилегий, назначенных используемой роли.

Оператор предоставления пользователю права использования роли имеет следующий синтаксис:

GRANT {<имя_роли1> [, <имя_роли2> ...]}
TO {PUBLIC | {[USER] <имя_пользователя1> [, [USER] <имя_пользователя2>...] } }
[WITH ADMIN OPTION];

Предложение WITH ADMIN OPTION позволяет пользователю, в свою очередь, назначать эту роль другим пользователям. Например, если пользователю USERA назначили роль DEVELOPER с предложением WITH ADMIN OPTION, то он может назначить эту роль другим пользователям. В следующем примере создается роль FULL_ACCESS, ей выдается привилегия ALL для таблицы TEST_SCORES, а затем роль назначается пользователю BJONES.

CREATE ROLE full_access;
GRANT all ON test_scores TO full_access;
GRANT full_access TO bjones;

Если пользователь BJONES попытается подключиться к БД без указания роли FULL_ACCESS, то он получит отказ в доступе к таблице TEST_SCORES, потому что только роли были выданы необходимые привилегии. Если BJONES указывает роль при подключении, то он получает полный доступ к таблице, т.к. роли FULL_ACCESS была выдана привилегия ALL для таблицы TEST_SCORES.

При подключении к базе данных пользователь может использовать роль, указав её в строке подключения. *Это единственная возможность, когда пользователь может указать роль*. InterBase не предоставляет возможности переключения между ролями, сохраняя подключение. *Пользователь должен отключиться и затем подключиться с новой ролью.*

Распространенная ошибка. Часто встречается неправильное представление о механизме ролей, что после того как пользователю назначили роль, он сразу же получает все привилегии, выданные этой роли. Это неверно. Чтобы пользователь приобрел все привилегии, назначенные роли, он должен указать эту роль при подключении к базе данных. Если же пользователь при подключении не указал имя роли, то ему не будут доступны ни одна из привилегий этой роли.

Задание для самостоятельной работы

1. Создайте роли ***R1*** и ***R2***.

2. Назначьте привилегии:

- роли ***R1*** - просматривать таблицы БД а также добавлять и удалять строки в таблицы;
- роли ***R2*** – полный доступ к БД.

3. Назначьте *каждую* роль *каждому* пользователю.

4. Проверьте работу приложения для всех комбинаций пользователей и ролей. При необходимости внесите изменения в проект для обеспечения его работоспособности. Сделайте выводы.

3.4. Отмена привилегий роли

Используя команду REVOKE, можно отменить привилегии для ролей, которым до этого были назначены соответствующие привилегии. Если какая-то привилегия отнимается у роли, то все пользователи, использующие эту роль, теряют право на выполнение соответствующих команд. Синтаксис команды REVOKE в данном случае следующий:

REVOKE <привилегия> ON [TABLE] <имя_таблицы> FROM <имя_роли>;

Пример отмены привилегии INSERT для таблицы TEST_SCORES у роли FULL_ACCESS.

REVOKE insert ON test_scores FROM full_access;

3.5. Отмена ролей пользователя

Имеется возможность как назначить роль пользователю, так и отменить право на ее использование. Если у пользователя отнимается право на использование роли, то он больше не сможет подключиться к базе данных с указанием этой роли и, соответственно, ему больше не доступны привилегии, назначенные этой роли. Синтаксис отмены роли пользователю:

REVOKE <имя_роли> FROM <имя_пользователя>;

В примере пользователю BJONES отменяется право на использование роли FULL_ACCESS. После чего BJONES больше не сможет подключиться с указанием этой роли.

REVOKE full_access FROM bjones;

3.6. Удаление роли

Когда роль перестает быть нужной, её можно удалить из базы данных. Удалить роль имеет право SYSDBA или пользователь, создававший роль. При удалении роли, все привилегии, назначенные ей, также удаляются из базы данных. Для удаления роли используется следующий синтаксис:

DROP ROLE <имя_роли>;

Пример демонстрирует удаление роли FULL_ACCESS:

DROP ROLE full_access;

Задание для самостоятельной работы

1. Отмените привилегию редактирования столбцов таблицы *GRUPPA* для роли ***R2*** и проверьте работу приложения.

ЛАБОРАТОРНАЯ РАБОТА №6
КЭШИРОВАНИЕ ИЗМЕНЕНИЙ

1. Кэширование изменений с помощью свойства CachedUpdates

Суть кэширования изменений заключается в том, что в каталоге запуска программы создается локальная копия данных и все последующие изменения относятся не к реальным данным таблиц БД, а к хранящейся в буфере (кэше) их локальной копии. После изменения данных в КЭШе они могут быть либо перенесены в реальные таблицы БД (подтверждение изменений), либо кэш ликвидируется без запоминания изменений (откат изменений).

Любой набор данных (НД) может быть переведен в режим кэширования изменений, если в его свойство ***CachedUpdates*** поместить значение **True**. По умолчанию это свойство равно **false** и кэширование не производится. Если установить его в **true**, то все изменения набора данных будут кэшироваться. Они передаются в базу данных только после выполнения метода **ApplyUpdates**. Если же после множества изменений выполнить метод **CancelUpdates**, то все изменения, произведенные после последнего выполнения **ApplyUpdates**, отменяются. Метод **CommitUpdates** очищает буфер кэша, после чего он готов для приема новой порции информации.

Помимо очевидных преимуществ кэширования изменений (снижение нагрузки на сеть и отсутствие блокировок данных в условиях многопользовательского доступа к ним), этот прием позволяет реализовать разного рода диалоговые окна (формы), в которых пользователь может вначале изменить данные, но потом передумать и отказаться от сделанных изменений.

ЗАДАНИЕ. Проверьте это, построив простое приложение (рис. 13) для работы с таблицей PERS базы данных IB_PROBA, созданной при выполнении лабораторной работы №4.

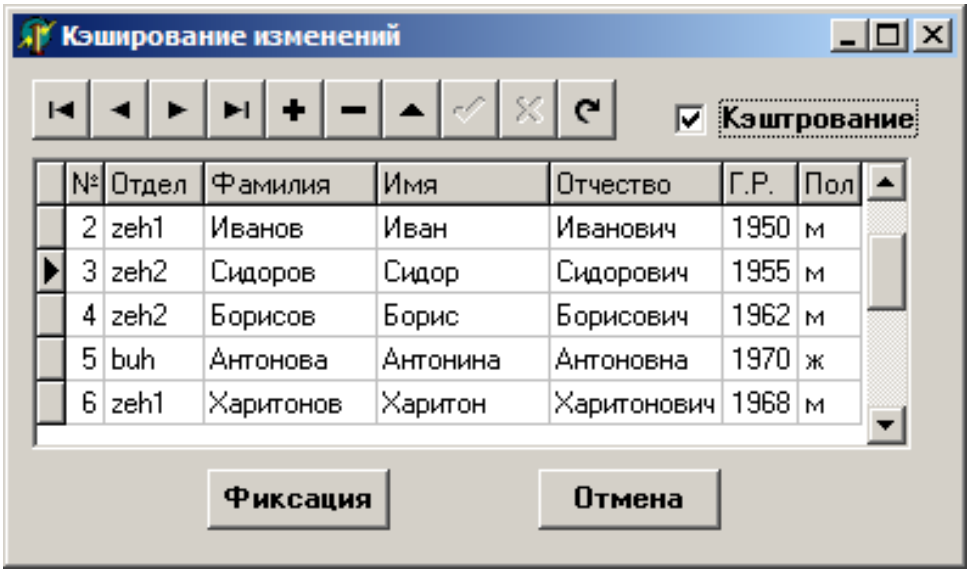

Рис.13. Приложение, демонстрирующее кэширование данных

Индикатор ***Кэширование*** переключает режим кэширования. Кнопка ***Фиксация*** фиксирует все сделанные изменения в базе данных. Кнопка ***Отмена*** отменяет сделанные изменения. Когда приложение закрывается, надо проверить, не работало ли оно в режиме кэширования, и не было ли сделано изменений, которые не зафиксированы в базе данных. Если были, то следует спросить пользователя о необходимости их сохранения и при положительном ответе зафиксировать изменения.

1. Для таблицы **Table1** свойство ***CachedUpdates*** первоначально задать равным **true**.

2. В приложение ввести переменную ***modif***, которая будет фиксировать наличие несохраненных изменений:

```
var modif: boolean = false;
```

3. Для события **OnClick** индикатора ***Кэширование*** предусмотреть обработчик:

```
Table1.CachedUpdates := not Table1.CachedUpdates;
BApplyUpdates.Enabled := Table1.CachedUpdates;
BCancelUpdates.Enabled := Table1.CachedUpdates;
if Table1.CachedUpdates then modif := false;
```

Он изменяет режим кэширования таблицы, делает доступными или недоступными в зависимости от режима кнопки ***Фиксация*** (ее имя — **BApplyUpdates**) и ***Отмена*** (ее имя — **BCancelUpdates**) и, если включается режим кэширования, то сбрасывается в **false** значение ***modif***.

4. Для события **AfterEdit** компонента **Table1** предусмотреть обработчик:

```
if Table1.CachedUpdates then modif := true;
```

который фиксирует в переменной ***modif*** факт редактирования записи.

5. Обработчик события **OnClick** кнопки ***Фиксация*** имеет вид:

```
Table1.ApplyUpdates;
Table1.CommitUpdates;
Modif := false;
```

6. Обработчик события **OnClick** кнопки ***Отмена*** имеет вид:

```
Table1.CancelUpdates;
modif := false;
```

В обоих обработчиках переменная ***modif*** сбрасывает в **false**, фиксируя отсутствие не сохраненных изменений.

7. Обработчик события **OnCloseQuery** формы имеет вид:

```
if Table1.CachedUpdates and modif
then
 case Application.MessageBox('Сохранить изменения в базе
  данных?', 'Подтвердите сохранение изменений',
  MB_YESNOCANCEL+MB_ICONQUESTION) of
  IDYES: Table1.ApplyUpdates;
  IDCANCEL: CanClose := false;
  IDNO: Table1.CancelUpdates;
 end;
```

При наличии не сохраненных изменений пользователю задается вопрос «Сохранить изменения в базе данных?». При положительном ответе изменения сохраняются методом **ApplyUpdates**. При отрицательном — изменения отменяются методом **CancelUpdates**. Если при получении запроса пользователь нажал кнопку **Cancel**, то приложение не закрывается (**CanClose** задается равным **false**).

8. Сделайте такое приложение и поработайте с ним, чтобы почувствовать различие в работе при наличии и отсутствии кэширования.

2. Кэширование изменений с помощью Query и UpdateSQL

Метод **ExecSQL** осуществляет немедленную модификацию таблицы. Однако нередко удобнее было бы кэшировать изменения (хранить их временно в памяти), а после того, как все изменения и проверки сделаны, переслать их в базу данных или, по решению пользователя, отменить все сделанные исправления. Это делается так же, как и для компонента **Table**: устанавливается в **true** свойство ***CachedUpdates*** компонента **Query** и применяются методы **AppIyUpdates** для записи изменений в базу данных, метод **CancelUpdates** для отмены изменений и метод **CommitUpdates** для очистки буфера кэша.

Но режим кэширования позволяет сделать большее — он позволяет подключить в приложение компонент **UpdateSQL**. Этот компонент, расположенный на странице библиотеки *BDE*, позволяет модифицировать наборы данных, открытые в режиме только для чтения. Это особенно важно для наборов данных, открываемых **Query** с запросом **Select**, поскольку **Select** создает таблицу только для чтения.

Задание для самостоятельной работы

Постройте приложение, демонстрирующее режим кэширования и тождественное тому, которое рассматривалось выше.

1. Возьмите то же самое приложение (рис. 13), только замените в нем компонент **Table1** на компонент **Query1**, и везде в тексте тоже проведите соответствующую замену **Table1** на **Query1**.

2. В свойстве **SQL** компонента **Query1** запишите «**Select * from Pers**» и установите свойство ***CachedUpdates*** в **true**.

3. Запустите свое приложение, и вы увидите, что оно, увы, не работает. Отредактировать запись или вставить новую запись невозможно. А из кнопок навигатора доступны только кнопки навигации. Причина всего этого была указана ранее — **Query** с запросом **Select** создает таблицу только для чтения.

4. Исправьте ваше приложение. Поместите на него компонент **UpdateSQL** со страницы библиотеки *BDE*. Чтобы связать его с приложением, установите в компоненте **Query1** свойство ***UpdateObject*** равным имени введенного компонента **UpdateSQL1**. Это имя вы можете выбрать из выпадающего списка в свойстве ***UpdateObject***.

5. Теперь можно задавать для компонента **UpdateSQL1** свойства ***DeleteSQL***, ***InsertSQL*** и ***ModifySQL***. Они содержат соответственно запросы, которые должны выполняться при удалении, вставке или модификации записи. Эти запросы можно записать обычным образом, вручную. А можно воспользоваться редактором UpdateSQL, который вызывается двойным щелчком на **UpdateSQL** или при выделенном **UpdateSQL** вызывается из всплывающего меню. Окно этого редактора имеет вид, представленный на рис. 14.

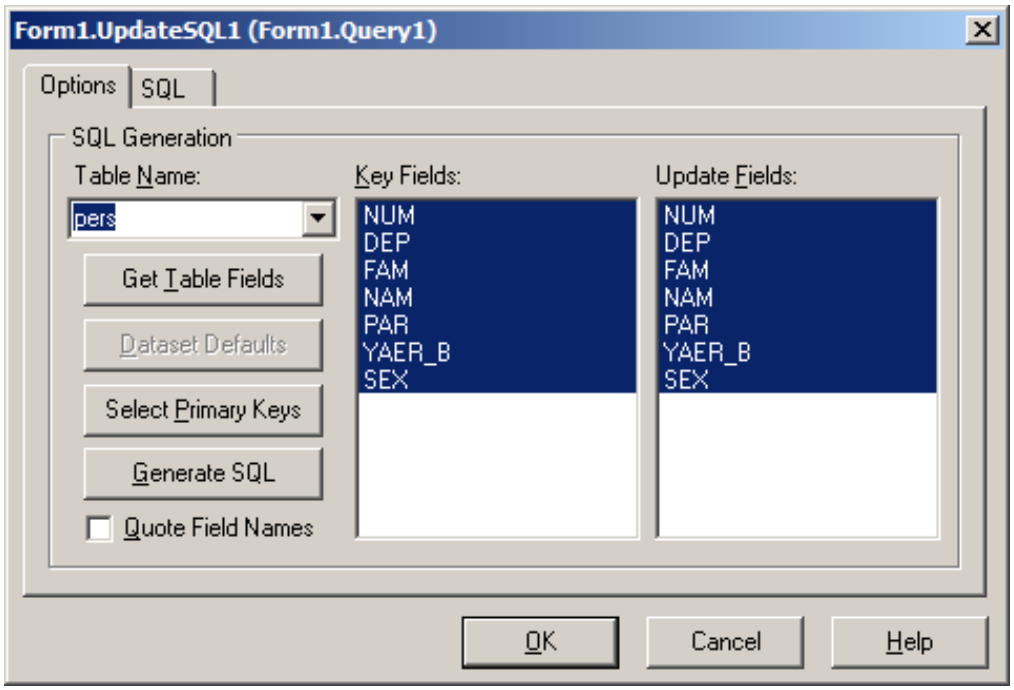

Рис.14. Окно редактора UpdateSQL

Первая страница ***Options*** Редактора UpdateSQL, представленная на рис. 14, содержит два окна *Key Fields* и *Update Fields*. В левом окне *Key Fields* надо выделить поля, по которым программа будет распознавать модифицируемую или удаляемую запись. Можно выделить все поля, что гарантирует максимально возможную надежность распознавания, но обычно следует ограничиться выбором нескольких наиболее важных полей. При работах с очень большими таблицами это сэкономит время выполнения запроса. Для выбора совокупности полей можно нажать клавишу Clrl и, не отпуская ее, выбрать курсором мыши нужные поля.

В правом окне *Update Fields* надо выделить поля, значения которых будут задаваться при модификации или вставке записи. Обычно целесообразно выделить в этом окне все поля, хотя, конечно, могут быть случаи, когда не все поля надо задавать (как мы покажем далее, у нас именно такой случай). После того как вы выделили в этих окнах поля, нажмите кнопку **Generate SQL**. После этого вам будет показана вторая страница ***SQL*** редактора UpdateSQL, на которой вы можете просмотреть сгенерированные запросы для модификации (Modify), вставки (Insert) и удаления (Delele) записи. После щелчка на **ОК** эти запросы перенесутся в свойства **DeleteSQL**, **InsertSQL** и **ModifySQL**, где вы их можете дополнительно отредактировать.

При всех выделенных полях запрос **ModifySQL** будет иметь вид:

```
UPDATE PERS
SET
  Num = :Numa,
  Dep = :Dep,
  Fam = :Fam,
  Nam = :Nan,
  Par = :Par,
  Year_b = :Year_b,
  Sex = :Sex
WHERE
  Num = :OLD_Num  and  Dep = :OLD_Dep and
  Fam = :OLD_Fam  and  Nam = :OLD_Nam and
  Par = :OLD_Par  and  Year_b = :OLD_Year_h and
  Sex = :OLD_Sex
```

В нем в разделе ***Set*** указана установка всех полей в значения, задаваемые соответствующими параметрами с именами, тождественными именам полей. В этот раздел включаются те поля, которые вы выделили в окне *Update Fields* редактора UpdateSQL. Заполнение этих параметров при выполнении соответствующих команд приложения вам не потребуется: все это сделают автоматически методы компонента UpdateSQL. В разделе ***Where*** содержатся условия, по которым идентифицируется модифицируемая запись. В этих условиях используются параметры с именами, тождественными именам полей, но с префиксом OLD. Эти параметры — прежние значения соответствующих полей, которые были получены компонентом до модификации записи. В условия ***Where*** вклю-

чены те поля, которые вы выделили в окне *Key Fields* редактора UpdateSQL, Естественно, что в зависимости от приложения вы можете заменить эти автоматически сгенерированные имена параметров на другие или вообще использовать запросы без параметров. Только имейте в виду, что значения автоматически сгенерированных параметров в дальнейшем будут автоматически загружаться из объектов-полей, а если вы от них откажетесь, то и соответствующие значения должны будете задавать сами.

Запросы в свойствах **DeleteSQL** и **InsertSQL** строятся по такому же принципу.

Теперь вернемся к нашему примеру. Вглядитесь в приведенный выше запрос **update**. В разделе ***Set*** в нем указана возможность обновления всех полей. Но для нашей базы данных это неприемлемо, так как значение поля **Num** задавать нельзя. Оно автоматически рассчитывается в таблице базы данных. Так что в правой панели окна редактора UpdateSQL (рис. 14) нужно выделить все поля, кроме поля **Num**. В разделе ***Where*** приведенного выше запроса **update** фигурируют все поля. Это значит, что при многопользовательском доступе к данным запись, которая должна измениться, будет идентифицироваться по всем полям.

Для нашего случая это неверно. Все поля, кроме поля **Num**, вряд ли разумно включать в этот список. А вдруг, пока один пользователь получил и редактирует запись, другой пользователь что-то в ней изменил. Например, сотрудница вышла замуж и сменила фамилию. Или сотрудника перевели в другой отдел. Но даже если исключить подобные мало правдоподобные ситуации, идентификация записи по множеству полей в нашем случае нецелесообразна. Она только увеличит время выполнения запросов, не повышая надежности работы.

В нашей таблице имеется поле **Num**, которое обеспечивает уникальность каждой записи. Так что в левой панели *Key Fields* окна редактора UpdateSQL (рис. 15) надо выделить только одно поле — **Num**.

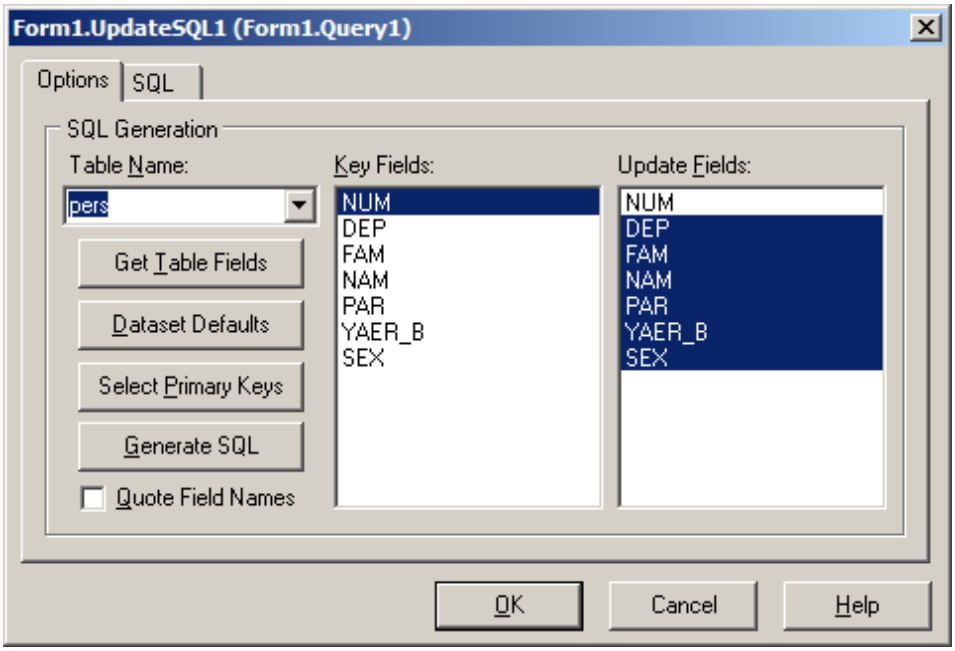

Рис.15. Окно редактора UpdateSQL с внесенными изменениями

При правильном выделении полей после щелчка на кнопке **Generate SQL** запрос ***ModifySQL*** будет иметь вид:

```
UPDATE PERS
SET
  Dep = :Dep,
  Fam = :Fam,
  Nam = :Nam,
  Par = :Par,
  Year_b = :Year_b,
  Sex = :Sex
WHERE
  Num = :OLD_Num
```

6. Запустите теперь приложение, и вы увидите, что оно стало работать так же, как работало ранее с компонентом **Table**. Можно редактировать записи, удалять, вставлять новые записи, управлять кэшированием.

В денном случае все получилось так просто, поскольку соответствующие методы работы с **UpdateSQL** автоматически выполняются компонентами **DBNavigator** и **DBGrid**. В некоторых других приложениях эти методы надо вызывать явно. Поэтому коротко ознакомимся с ними.

Метод **Apply(UpdateKind: TUpdateKind)** обеспечивает загрузку значений всех необходимых параметров и выполнение одного из запросов компонента UpdateSQL. Какого именно — определяется параметром UpdateKind, который может принимать значения ***ukModify***, ***ukInsert*** или ***ukDelete***, что соответствует выполнению запроса, хранящегося в свойствах *ModifySQL*, *InsertSQL* и *DeleteSQL*.

Например, оператор UpdateSQL1.Apply (ukModify) вызовет выполнение запроса, содержащегося в свойстве ModifySQL.

Если нужный запрос SQL не содержит параметров, то эффективнее вызвать метод **ExecSQL(UpdateKind: TUpdateKind)**, который тоже обеспечивает выполнение указанного запроса, но без предварительной загрузки значений параметров.

Наконец, метод **SetParams(UpdateKind: TUpdateKind)** обеспечивает загрузку значений параметров указанного запроса SQL без его выполнения. Таким образом, метод Apply — это просто последовательное выполнение методов SetParams и ExecSQL.

Список литературы

1. Архангельский А.Я. Программирование в Delphi 5. - М.: БИНОМ, 2000. - 1072 с.

2. Архангельский А.Я. Программирование в Delphi 7. – М.: ООО «Бином-Пресс», 2003 г. – 1152 с.

3. Ковязин А., Востриков С. Мир InterBase. Архитектура, администрирование и разработка приложений баз данных в InterBase/Firebird/Yaffil. - 2-е изд., доп. – М.: КУДИЦ-ОБРАЗ, 2002. – 496 с.

4. Фаронов В.В. Программирование баз данных в Delphi 7: Учебный курс. – СПб.: Питер, 2006. – 459 с.

5. Шумаков П.В. Delphi 3 и разработка приложений баз данных. – М.: НОЛИДЖ, 1998. - 704 с.

СОДЕРЖАНИЕ

ЛАБОРАТОРНАЯ РАБОТА №4 УТИЛИТА DATABASE EXPLORER 3
ЛАБОРАТОРНАЯ РАБОТА №5 ЗАЩИТА ТАБЛИЦ СЕРВЕРА INTERBASE 10
1. Управление пользователями 10
2. Предоставление привилегий 12
2.1. Оператор GRANT 13
2.2. Оператор REVOKE 14
3. Использование ролей 15
3.1. Создание роли 15
3.2. Предоставление привилегий роли 16
3.3. Назначение роли пользователю 16
3.4. Отмена привилегий роли 17
3.5. Отмена ролей пользователя 17
3.6. Удаление роли 18
ЛАБОРАТОРНАЯ РАБОТА №6 КЭШИРОВАНИЕ ИЗМЕНЕНИЙ 18
1. Кэширование изменений с помощью свойства CachedUpdates 18
2. Кэширование изменений с помощью Query и UpdateSQL 20

Тетюшева Светлана Геннадьевна

Базы данных «КЛИЕНТ – СЕРВЕР»

Методические рекомендации

для студентов специальностей 010101, 050202

(часть II)

Редактор Н.М. Устюгова

……..……
Подписано в печать __.__.09 Формат 60х84 1/16 Бумага тип. №1
Печать трафаретная Усл.печ.л. 1,75 Уч.-изд.л. 1,75
Заказ № Тираж 50 Цена свободная
……..…
РИЦ Курганского государственного университета.
640669, г. Курган, ул. Гоголя, 25.
Курганский государственный университет.