МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ

ГОСУДАРСТВЕННОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ
ВЫСШЕГО ПРОФЕССИОНАЛЬНОГО ОБРАЗОВАНИЯ
НИЖЕГОРОДСКИЙ ГОСУДАРСТВЕННЫЙ
ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ им. Р.Е.Алексеева

Кафедра «Информатика и системы управления»

БАЗЫ ДАННЫХ

СТРУКТУРИРОВАННЫЙ ЯЗЫК ЗАПРОСОВ И ГЕНЕРАТОР ОТЧЕТОВ

Методические указания к лабораторной работе №4 для студентов специальности 230102, 230201

Нижний Новгород 2010

Составитель Балашова Т.И.

ББК

БАЗЫ ДАННЫХ. СТРУКТУРИРОВАННЫЙ ЯЗЫК ЗАПРОСОВ И ГЕНЕРАТОР ОТЧЕТОВ: учебно-методические указания к лабораторной работе №4 для студентов специальности 230102, 230201/ НГТУ; сост.: Балашова Т.И. Н.Новгород 2010. 30с.

В учебно-методические материалах дано описание использования структурированного языка запросов и генератора отчетов на примере разработки базы данных с использованием Borland Delphi 7 и Rave Reports 5

Подписано к печати . Формат 60x84 1/16. Бумага офсетная.
Печать офсетная. Усл.печ.л. Уч.-изд. л. .Тираж экз. Заказ .

Нижегородский государственный технический университет
им. Р.Е. Алексеева
Типография НГТУ им. Р.Е. Алексеева
Адрес университета и полиграфического предприятия:
603950, Нижний Новгород, ул. Минина, 24.

Цель работы: изучить структурированный язык запросов.

КРАТКИЕ СВЕДЕНИЯ ИЗ ТЕОРИИ

1. ВВЕДЕНИЕ

Аббревиатура SQL символизирует собой *Структурированный Язык Запросов*. SQL – это язык, который дает возможность работать в реляционных базах данных, которые являются наборами связанной информации, хранимой в таблицах.

Независимость от специфики компьютерных технологий, а также его поддержка лидерами промышленности в области технологий реляционных баз данных сделали SQL и, вероятно, в течение обозримого будущего оставит его основным стандартным языком.

Стандарт SQL определяется ANSI (*Американским Национальным Институтом Стандартов*) и в данное время также принимается ISO (*Международной Организацией по Стандартизации*).

SQL не изобретался ANSI. По существу, это изобретение IBM. Однако SQL быстро подхватили другие компании. По крайней мере, одна из них (Oracle) отбила у IBM право на рыночную продажу SQL продуктов. После того, как на рынке появился ряд конкурирующих SQL – программ, ANSI определил стандарт, к которому они должны были быть приведены (определение таких стандартов и является функцией ANSI). Однако после этого появились некоторые проблемы. Возникли они в результате стандартизации ANSI, которая вносила ряд ограничений. Так как ANSI не всегда определяет то, что является наиболее полезным, программы пытались соответствовать этому стандарту, не позволяя ему ограничивать их слишком сильно. Это, в свою очередь, привело к некоторому разнообразию.

Поэтому, прежде чем приступить к работе с SQL, необходимо ознакомиться с документацией вашего пакета программ, который вы будете использовать, чтобы знать, где в нем этот стандарт видоизменен.

2. КАК РАБОТАЕТ SQL

SQL — это язык, специально ориентированный на работу с реляционными базами данных. Он резко сокращает объем работы, которую вы должны были бы сделать, используя универсальный язык программирования, такой как C. Чтобы создать реляционную базу данных на C, вам потребовалось бы определить объект, называемый таблицей, которая могла бы расти, а затем постепенно определять процедуры для работы с ней. Например, для поиска записей из этой таблицы, вам необходимо было бы по шагам выполнить следующую процедуру:

1. Рассмотреть записи таблицы.

2. Выполнить проверку – является ли эта запись одной из записей, которые нам нужны.
3. Если это так, необходимо ее где – нибудь сохранить, пока не будет проверена вся таблица.
4. Проверить, имеются ли другие записи, удовлетворяющие критерию поиска.
5. Если имеются, вернуться на шаг 1.
6. Если записей больше нет, то вывести все значения, сохраненные в шаге 3 (если они вообще есть).

(Конечно, это не фактический набор команд С, а только логика шагов, которые должны были бы быть выполнены в реальной программе.)

SQL сэкономит ваше время. Команды в SQL могут работать со всеми группами таблиц как с единым объектом и могут обрабатывать любое количество информации, полученной из них в виде единого модуля.

2.1. ИНТЕРАКТИВНЫЙ И ВЛОЖЕННЫЙ SQL

SQL бывает двух видов: интерактивный и вложенный. По большей части обе формы работают одинаково, но используются по – разному.

Интерактивный SQL используется непосредственно в базе данных для вывода результата и дальнейшего его использования пользователем. В этой форме ввод команд SQL производится самим пользователем.

Вложенный SQL состоит из команд, помещенных внутри программ, которые обычно написаны на другом языке (типа Кобола или Паскаля). Это делает программы более мощными и эффективными. Однако приходится иметь дело со структурой SQL и стилем управления данными, который требует некоторых расширений к интерактивному SQL.

2.2. ТИПЫ ДАННЫХ

Типы значений, которые могут занимать поля таблицы, не являются логически одинаковыми. Наиболее очевидное различие – между числами и текстом. Например, нельзя помещать числа в алфавитном порядке или вычитать одно имя из другого.

Так как системы реляционных баз данных основаны на связях между фрагментами информации, различные типы данных должны объективно отличаться друга от друга, чтобы над ними можно было выполнить соответствующие операции.

В SQL это делается с помощью назначения каждому полю типа данных, который указывает, какое значение это поле может содержать. Все значения в данном поле должны иметь одинаковый тип. Это ограничение очень удобно, так как оно налагает некоторую структурность на данные.

К сожалению, определение этих типов является основной областью, в которой большинство коммерческих программ баз данных и официальный стандарт SQL не всегда совпадают. Стандарт ANSI подразумевает только

текст и тип номера, в то время как большинство коммерческих программ используют другие специальные типы, такие как **DATA** (ДАТА) и **TIME** (ВРЕМЯ) – уже практически стандартные типы данных (хотя их точный формат меняется). Некоторые пакеты также поддерживают такие типы, как **MONEY** (ДЕНЬГИ) и **BINARY** (ДВОИЧНЫЙ).

ANSI определяет несколько различных типов значений чисел, различия между которыми довольно тонки и иногда их путают. Разрешенные ANSI типы данных перечислены в Прил. А.

Два числовых типа ANSI**, INTEGER** (ЦЕЛОЕ ЧИСЛО) и **DECIMAL** (ДЕСЯТИЧНОЕ ЧИСЛО) будут предпочтительными для целей большинства прикладных бизнес – программ.

Текстовый тип данных — **CHAR** (СИМВОЛ) относится к строке текста. Поле типа CHAR имеет определенную длину, которая задается максимальным числом символов, вводимых поле.

Больше всего реализаций имеет нестандартный тип, называемый **VARCHAR** (ПЕРЕМЕННОЕ ЧИСЛО СИМВОЛОВ), который является текстовой строкой. Различие между CHAR и VARCHAR заключается в том, что CHAR сразу резервирует столько памяти, сколько необходимо для хранения максимального количества символов (даже если максимум достигнут не будет), а VARCHAR резервирует только то количество памяти, которое необходимо для хранения введенного в поле значения.

Символьные типы состоят из всех печатных символов, включая числа, и сохраняются в компьютере как двоичные значения, но для пользователя представляются как печатный текст.

В отношении таких типов как DATE мы должны следить за рынком, а не за ANSI. Рекомендуется ознакомиться с документацией вашего пакета программ, чтобы выяснить, какие типы данных поддерживаются.

2.3. КЛЮЧЕВЫЕ СЛОВА И ТЕРМИНОЛОГИЯ

SQL имеет специальные термины, которые используются для его описания. Среди них такие слова как *запрос*, *предложение* и *предикат*, которые являются важнейшими в описании и понимании языка, но не означают что – нибудь самостоятельное для SQL.

Ключевые слова — это слова, которые имеют специальное значение в SQL. Они могут быть командами и пишутся чаще всего прописными буквами. Команды, или предложения, являются инструкциями, которыми вы обращаетесь к базе данных. Команды состоят из одной или более отдельных логических частей, называемых *предложениями*. Предложения начинаются с ключевого слова. Например, «SELECT sname», «FROM Salespeope» и «WHERE city = London». *Аргументы* завершают или изменяют значение предложения.

В приведенном примере, «Salespeople» — аргумент, а «FROM» — ключевое слово предложения FROM. Аналогично, «city = London» — аргумент предложения WHERE.

Объекты – структуры базы данных, хранимые в памяти, которым даны имена. Они включают в себя базовые таблицы, представления и индексы.

3. ИСПОЛЬЗОВАНИЕ SQL ДЛЯ ИЗВЛЕЧЕНИЯ ИНФОРМАЦИИ ИЗ ТАБЛИЦ

Как подчеркивалось ранее, SQL символизирует собой структурированный язык запросов. Запросы, вероятно, наиболее часто используемый аспект SQL. Фактически для категории SQL пользователей маловероятно, чтобы кто – либо использовал этот язык для чего – то другого. По этой причине начнем обсуждение SQL с обсуждения понятия запроса и как он выполняется на этом языке.

Запрос — команда, которую вы даете вашей программе базы данных и которая сообщает ей, чтобы она вывела определенную информацию из таблиц в память. Эта информация обычно посылается непосредственно на экран компьютера или терминала, которым вы пользуетесь, хотя, в большинстве случаев, ее можно также послать принтеру, сохранить в файле (как объект в памяти компьютера) или представить как входную информацию для другой команды или процесса.

Все запросы в SQL состоят из одиночной команды. Структура этой команды обманчиво проста, потому что вы должны расширять ее так, чтобы выполнить сложную обработку данных. Эта команда называется — **SELECT.**

3.1. КОМАНДА SELECT

В самой простой форме, команда SELECT просто инструктирует базу данных, чтобы извлечь информацию из таблицы. Например, вы могли бы вывести таблицу «Продавцов», напечатав следующее:

```
SELECT snum, sname, sity, comm
FROM Salespeople.
```

Результат выполнения данного запроса будет выглядеть так:

```
===============  SQL Execution Log ============
| SELECT snum, sname, sity, comm                 |
| FROM Salespeople;                              |
| ===============================================|
|   snum      sname          city          comm  |
| ------    ----------   -----------   -------  |
|   1001      Peel         London        0.12   |
|   1002      Serres       San Jose      0.13   |
|   1004      Motika       London        0.11   |
|   1007      Rifkin       Barcelona     0.15   |
|   1003      Axelrod      New York      0.10   |
 ================================================
```

В данном примере слово SELECT является ключевым словом, которое сообщает базе данных, что эта команда – запрос. Все запросы начинаются этим словом, сопровождаемым пробелом.

Слова snum, sname, sity, comm являются списком столбцов, которые выбираются запросом. FROM – ключевое слово подобно SELECT, которое также должно присутствовать в каждом запросе. Оно сопровождается пробелом и именем таблицы, из которой производится выбор.

Символ точки используется для указания конца запроса. Символ конца запроса может отличаться в различных системах.

Для вывода всех данных всех столбцов таблицы вовсе не обязательно перечислять все столбцы после SELECT. Для этого можно воспользоваться символом * после SELECT. При этом на результате это никак не отразится.

3.2. УСТРАНЕНИЕ ИЗБЫТОЧНОСТИ ВЫВОДИМЫХ ДАННЫХ

Для наглядности рассмотрим ситуацию, когда необходимо вывести номера всех продавцов, которые участвовали в продажах за прошедший день. При этом может получиться так, что какой – либо продавец продал несколько товаров и фигурирует в записях неоднократно. Чтобы вывести номера продавцов независимо от того, сколько раз они встречаются в таблице, применяется специальное слово DISTINCT (ОТЛИЧИЕ). Инструкция:

```
SELECT DISTINCT snum
FROM Orders.
```

выведет каждый номер в единственном экземпляре. При выборе нескольких столбцов с использованием предложения DISTINCT будут выведены только неповторяющиеся записи.

Вместо DISTINCT можно указать предложение ALL, которое обеспечит вывод всех записей из таблицы. Использование ALL равносильно случаю, когда вы не указываете ни DISTINCT ни ALL и носит скорее пояснительный, чем действующий характер.

3.3. КВАЛИФИКАЦИОННЫЙ ВЫБОР

С течением времени таблицы имеют тенденцию становиться все больше и больше. Поскольку в большинстве случаев пользователя интересуют не все записи, SQL предоставляет возможность устанавливать критерий, по которому будут выбираться записи. Для этого в SQL есть специальное предложение WHERE, которое позволяет ставить условия, истинные или ложные для каждой записи таблицы.

Например, команда

```
SELECT sname, city
FROM Salespeople
WHERE city = "LONDON".
```

выведет имена всех продавцов, которые проживают в Лондоне. При квалификационном выборе наиболее важно то, что вы можете ставить условие, называемое *предикатом*, которое определяет или не определяет указанную строку таблицы из тысяч таких же строк, будет ли она выбрана для вывода. Предикаты могут становиться очень сложными, предоставляя вам высокую точность в решении. Именно эта способность решать точно, что вы хотите видеть, делает запросы SQL такими мощными.

4. ИСПОЛЬЗОВАНИЕ РЕЛЯЦИОННЫХ И БУЛЕВЫХ ОПЕРАТОРОВ ДЛЯ СОЗДАНИЯ БОЛЕЕ СЛОЖНЫХ ПРЕДИКАТОВ

4.1. РЕЛЯЦИОННЫЕ ОПЕРАТОРЫ

Реляционный оператор — математический символ, который указывает на определенный тип сравнения между двумя значениями.

Реляционные операторы, которыми располагает SQL:

=	*Равный*
>	*Больше чем*
<	*Меньше чем*
>=	*Больше чем или равно*
<=	*Меньше чем или равно*
<>	*Не равно*

Эти операторы имеют стандартные значения для числовых значений. Для значения символа, их определение зависит от формата преобразования **ASCII** или **EBCDIC**, который вы используете.

Использование приведенных выше операторов допускается не только к числовым значениям, но также и к символам. При этом SQL сравнивает символьные значения в терминах основных номеров, как определено в формате преобразования. В ASCII все символы верхнего регистра меньше, чем все символы нижнего регистра, поэтому «Z» < «a», а все номера меньше, чем все символы, поэтому «1» < «Z». То же относится и к EBCDIC.

4.2. БУЛЕВЫ ОПЕРАТОРЫ

Основные Булевы операторы также распознаются в SQL. Выражения Буля — являются или верными или неверными, подобно предикатам. Булевы операторы связывают одно или более верных/неверных значений и производят единственное верное/или/неверное значение. Стандартными операторами Буля, распознаваемыми в SQL, являются: **AND**, **OR**, и **NOT**.

Существуют другие, более сложные, операторы Буля (типа **XOR)**, но они могут быть сформированы из этих трех простых операторов – AND, OR, NOT. Булева логика основана на цифровой компьютерной операции и

фактически весь SQL (или любой другой язык) может быть сведен до уровня Булевой логики.

ОПЕРАТОРЫ БУЛЯ И КАК ОНИ РАБОТАЮТ:

AND берет два Буля (в форме A AND B) как аргументы и оценивает их по отношению к истине, верны ли они оба.

OR берет два Буля (в форме A OR B) как аргументы и оценивает на правильность, верен ли один из них.

NOT берет одиночный Булев (в форме NOT A) как аргументы и меняет его значение с неверного на верное, или верное на неверное.

5. ИСПОЛЬЗОВАНИЕ СПЕЦИАЛЬНЫХ ОПЕРАТОРОВ В УСЛОВИЯХ

В дополнение к реляционным и булевским операторам, обсуждаемым в пункте 4, SQL использует специальные операторы **IN, BETWEEN, LIKE,** и **IS NULL.**

5.1. ОПЕРАТОР IN

Оператор IN определяет набор значений, в которое данное значение может или не может быть включено. Для нахождения всех продавцов, которые проживают в Barcelona или в London, можно использовать следующий запрос:

```
SELECT *
FROM Salespeople
WHERE city = 'Barcelona' OR city = 'London'.
```

С использованием IN приведенный выше запрос будет выглядеть более упрощено:

```
SELECT *
FROM Salespeople
WHERE city IN ('Barcelona', 'London').
```

5.2. ОПЕРАТОР BETWEEN

Оператор **BETWEEN** похож на оператор IN. В отличие от определения по номерам из набора, как это делает IN, BETWEEN определяет диапазон между начальным и конечным аргументом, записываемым через AND. В отличие от IN, BETWEEN чувствителен к порядку, и первое значение в предложении должно быть первым по алфавитному или числовому порядку.

```
SELECT *
FROM Salespeople
WHERE comm BETWEEN .10 AND .12.
```

Приведенный пример выведет всех продавцов, чьи комиссионные лежат в диапазоне от 10 до 12. Также, подобно реляционным операторам, BETWEEN может работать с символьными полями в терминах эквивалентов ASCII. Это означает, что вы можете использовать BETWEEN, чтобы выбирать ряд значений из упорядоченных по алфавиту значений.

```
SELECT *
FROM Customers
WHERE cname BETWEEN 'A' AND 'G'.
```

Представленный выше запрос выведет всех заказчиков, первая буква в именах которых попала в определенный BETWEEN диапазон.

5.3. ОПЕРАТОР LIKE

LIKE применим только к полям типа CHAR или VARCHAR, с которыми он используется, чтобы находить подстроки. В качестве условия он использует групповые символы (wildkards) — специальные символы, которые могут соответствовать чему – нибудь. Имеются два типа групповых символов используемых с LIKE:

• символ подчеркивания (_) замещает любой одиночный символ. Например, '**b_t**' будет соответствовать словам '**bat**' или '**bit**', но не будет соответствовать'brat'.

• знак процента (**%**) замещает последовательность любого числа символов (включая символы нуля). Например, '**%p%t**' будет соответствовать словам '**put', 'posit',** или '**opt',** но не '**spite**'.

5.4. NULL ОПЕРАТОР

Довольно часто в таблицах будут встречаться записи, которые не имеют никаких значений для каждого поля, например, потому что информация не известна, или потому что это поле просто не заполнялось. SQL учитывает такой вариант, позволяя вам вводить значение **NULL** (ПУСТОЙ) в поле, вместо значения. Когда значение поля равно NULL, это означает, что программа базы данных специально промаркировала это поле как не имеющее никакого значения для этой строки (или записи). Это отличается от просто назначения полю значения нуля или пробела, которые база данных будет обрабатывать так же, как и любое другое значение. Точно так же, как NULL не является техническим значением, оно не имеет и типа данных. Оно может помещаться в любой тип поля. Тем ни менее, NULL в SQL часто упоминается как *нуль*. По этой причине SQL предоставляет специальный оператор **IS**, который используется с ключевым словом NULL для размещения значения NULL.

6. ОБОБЩЕНИЕ ДАННЫХ С ПОМОЩЬЮ АГРЕГАТНЫХ ФУНКЦИЙ

6.1. ЧТО ТАКОЕ АГРЕГАТНЫЕ ФУНКЦИИ?

Запросы могут производить обобщенное групповое значение полей точно так же, как и значение одного поля. Это делают с помощью агрегатных функций. Агрегатные функции производят одиночное значение для всей группы таблицы.

• **COUNT** производит номера строк или не – NULL значения полей, которые выбрал запрос.

• **SUM** производит арифметическую сумму всех выбранных значений данного поля.

• **AVG** производит усреднение всех выбранных значений данного поля.

• **MAX** производит наибольшее из всех выбранных значений данного поля.

• **MIN** производит наименьшее из всех выбранных значений данного поля.

Например,

```
SELECT SUM(amt)
FROM Orders;
```

Выведет сумму всех покупок.

6.2. ПРЕДЛОЖЕНИЕ GROUP BY

Предложение GROUP BY позволяет группировать сходные по нескольким полям записи с применением агрегатных функций. Например, для каждого продавца в таблице заказов необходимо вывести сумму всех его продаж:

```
SELECT snum, sname, SUM (amt)
FROM Orders
GROUP BY snum.
```

6.3. ПРЕДЛОЖЕНИЕ HAVING

Предложение HAVING используется для установки определенного критерия вывода (условия) подобно WHERE, но применяется в отличие от WHERE для групп записей. Например, необходимо вывести продавцов, чьи продажи превысили 30 000:

```
SELECT snum, sname, SUM (amt)
FROM Orders
```

```
GROUP BY snum
HAVING SUM (amt) > 30000.
```

7. ЗАПРОС ДЛЯ ЛЮБОГО ЧИСЛА ТАБЛИЦ С ПОМОЩЬЮ ОДНОЙ КОМАНДЫ

Одна из наиболее важных особенностей запросов SQL — это их способность определять связи между многочисленными таблицами и выводить информацию из них с учетом этих связей. Этот вид операции называется — *объединением*, которое является одним из видов операций в реляционных базах данных.

7.1. СОЗДАНИЕ ОБЪЕДИНЕНИЯ

Для примера рассмотрим ситуацию, когда необходимо найти для продавца его заказчиков, живущих в том же городе:

```
SELECT Customers.cname, Salespeople.sname, Salespeople.city
FROM Salespeople, Customers
WHERE Salespeople.city = Customers.city.
```

В приведенном примере, так как поле city содержится в обеих таблицах, имена таблиц будут использоваться как префиксы.

7.2. ОБЪЕДИНЕНИЕ ТАБЛИЦ ЧЕРЕЗ СПРАВОЧНУЮ ЦЕЛОСТНОСТЬ

Эта особенность часто используется просто для эксплуатации связей, встроенных в базу данных. В предыдущем примере мы установили связь между двумя таблицами в объединении. Но эти таблицы уже были соединены через snum поле. Эта связь называется *состоянием справочной целостности*. Используя объединение, можно извлекать данные в терминах этой связи. Например, чтобы показать имена всех заказчиков соответствующих продавцам, которые их обслуживают, можно использовать такой запрос:

```
SELECT Customers.cname, Salespeople.sname
FROM Customers, Salespeople
WHERE Salespeople.snum = Customers.snum.
```

7.3. ОБЪЕДИНЕНИЕ БОЛЕЕ ДВУХ ТАБЛИЦ

Также можно создавать запросы, объединяющие более двух таблиц. Предположим, что мы хотим найти все заказы клиентов, не находящихся в тех городах, где находятся их продавцы.

```
SELECT onum, cname, Orders.cnum, Orders.snum
FROM Salespeople, Customers,Orders
WHERE Customers.city < > Salespeople.city
```

```
AND Orders.cnum = Customers.cnum
AND Orders.snum = Salespeople.snum;
```

8. ИСПОЛЬЗОВАНИЕ ПОДЗАПРОСОВ

С помощью SQL можно вкладывать запросы внутрь друг друга. Обычно внутренний запрос генерирует значение, которое проверяется в предикате внешнего запроса, определяющего верно оно или нет.

Например, предположим, что мы знаем имя продавца: Motika, но не знаем значение его поля snum и хотим извлечь все заказы из таблицы заказов.

```
SELECT *
FROM Orders
WHERE snum = (SELECT snum
FROM Salespeople
WHERE sname = 'Motika').
```

Чтобы выполнить внешний (основной) запрос, SQL сначала должен выполнить внутренний запрос (или подзапрос) внутри предложения WHERE. В подзапросе производится поиск в таблице продавцов, где поле sname равно значению Motika, и затем извлекает значения поля snum для этих записей. Однако SQL не просто выдает это значение, а помещает его в предикат основного запроса вместо самого подзапроса.

```
WHERE snum = 1004
```

Основной запрос затем выполняется как обычно с вышеупомянутыми результатами.

9. ОПЕРАТОР EXISTS

Оператор EXISTS относится к тем операторам, которые принимают в качестве аргумента подзапрос. Сам оператор возвращает верное или неверное значение. Верное значение возвращается в том случае, если подзапрос производит какой – либо вывод. В остальных случаях его значение ложно. Иначе говоря, EXISTS – это выражение Буля. Рассмотрим работу данного оператора на примере. Предположим, перед нами стоит задача об извлечении данных о заказчиках, при условии, что хотя бы один из них проживает в San Jose.

```
SELECT cnum, cname, city
FROM Customers
WHERE EXISTS ( SELECT *
FROM Customers
WHERE city = " San Jose' ).
```

Внутренний запрос выбирает все данные для всех заказчиков в San Jose. Оператор EXISTS во внешнем предикате отмечает, был ли произведен

некоторый вывод подзапросом, и поскольку выражение EXISTS было полным предикатом, делает предикат верным.

10. ОБЪЕДИНЕНИЕ НЕСКОЛЬКИХ ЗАПРОСОВ

Можно поместить многочисленные запросы вместе и объединить их вывод, используя предложение UNION. Предложение UNION объединяет вывод двух или более SQL запросов в единый набор строк и столбцов. Например, чтобы получить всех продавцов и заказчиков, размещенных в Лондоне, и вывести их как единое целое, вы могли бы ввести:

```
SELECT snum, sname
FROM Salespeople
WHERE city = 'London'
UNION
SELECT cnum, cname
FROM Customers
WHERE city = 'London'.
```

***Примечание**: в данном примере количество полей и их типы для обоих объединяемых запросов одинаковые.*

ПРИМЕР ПРАКТИЧЕСКОЙ РЕАЛИЗАЦИИ

11. Выбор информации из базы данных

При работе с базой данных пользователя, как правило, интересует не все ее содержимое, а некоторая конкретная информация. Найти нужные сведения можно последовательным просмотром записей. Однако такой способ поиска неудобен и малоэффективен.

Большинство систем управления базами (СУБД) данных позволяют произвести выборку нужной информации путем выполнения запросов. Пользователь в соответствии с определенными правилами формулирует запрос, указывая, каким критериям должна удовлетворять интересующая его информация, а система выводит записи, удовлетворяющие запросу.

Для выборки из базы данных записей, удовлетворяющих некоторому критерию, предназначен компонент Query.

Компонент Query похож на компонент Table, но, в отличие от последнего, он представляет не всю базу данных (все записи), а только ее часть — записи, удовлетворяющие критерию запроса.

Для записи запроса в свойство SQL во время разработки формы используется редактор списка строк (String List Editor), окно которого открывается в результате щелчка на кнопке с тремя точками в строке свойства SQL окна инспектора объектов (Object Inspector) (рис. 1).

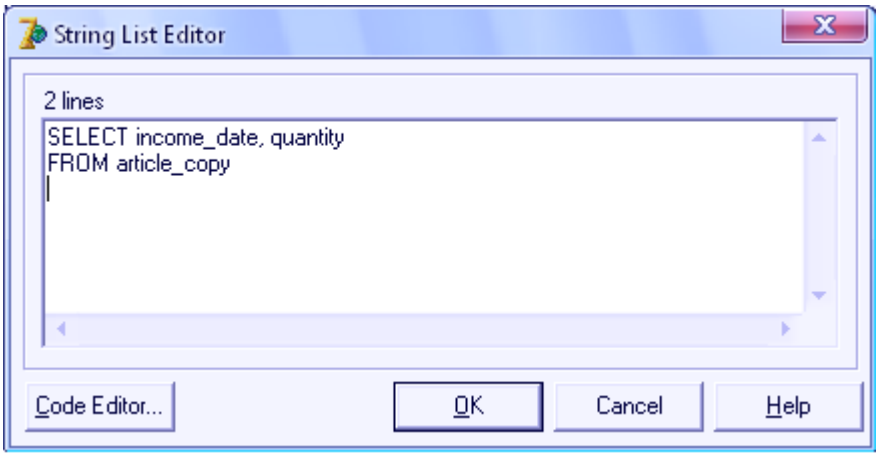

Рис. 1. Окно редактора списка строк

Для просмотра базы данных и результата выполнения запроса используется компонент DBGrid, который через компонент DataSource взаимодействует с компонентом Query.

Добавим на форму компоненты DataSource, Query и DBChart. Определим свойство DataBaseName как имя псевдонима БД, а свойство SQL зададим при помощи упомянутого ранее редактора строк. В окно редактора поместим следующий запрос:

```
SELECT DISTINCT Article.NAME ARTICLE, Article_type.NAME TYPE,
Vendor.NAME VENDOR, Shops.NAME SHOP, Shops.ADDRESS,
article_copy.QUANTITY, article_copy.INCOME_DATE
FROM article_copy
  INNER JOIN "article_vendor.DB" Article_vendor
  ON  (Article_vendor.ID = article_copy.ARTICLE_VENDOR_ID)
  INNER JOIN "shops.DB" Shops
  ON  (Shops.ID = article_copy.SHOP_ID)
  INNER JOIN "article.DB" Article
  ON  (Article.ID = Article_vendor.ARTICLE_ID)
  INNER JOIN "vendor.DB" Vendor
  ON  (Vendor.ID = Article_vendor.VENDOR_ID)
  INNER JOIN "article_type.DB" Article_type
  ON  (Article_type.ID = Article.ARTICLE_TYPE_ID)
GROUP BY Article_type.NAME, Article.NAME, Vendor.NAME,
Shops.NAME, Shops.ADDRESS, article_copy.QUANTITY,
article_copy.INCOME_DATE
HAVING  SUM( article_copy.QUANTITY ) BETWEEN 6 AND 10
ORDER BY ARTICLE
```

Изменим свойство Active на TRUE. Если все сделано верно, то свойство установится без предупреждений.

Далее определим свойства для компонента DataSource. Установим свойство DataSet как имя компонента Query.

После определения источника и набора записей, определим, через что будет выведена полученная в результате запроса информация. Для этого

добавим на форму DBGrid и укажем свойство DataSource как имя компонента DataSource.

Если все сделано верно, то в таблице появится результат запроса.

Теперь необходимо решить, как эргономичнее расположить таблицу с результатом запроса. Эргономичным вариантом, как и в случае DBChart, будет использование отдельной формы. Для показа результата запроса, добавим на форму кнопку и «под ней», в процедуре OnButtonClick, добавим следующий код:

```
dbgrd.Visible:=True; /dbgrd – имя формы для Grid
```

Создадим новую форму и поместим на нее компонент DBGrid. После этого, необходимо сделать компонент DBGrid увеличиваемым, т.е. чтобы его размер изменялся пропорционально размерам формы. Как и в случае с DBChart, определим несколько дополнительных строк в процедуре OnFormResize:

```
dbgrd.Width:=article_copy_form.Width;
dbgrd.Height:=article_copy_form.Height;
```

Это заставит DBChart «прилипнуть» к краям формы.

Примечание: *пример использования запросов продемонстрирован на компакт – диске в каталоге DBQuery.*

12. КОМПОНЕНТЫ RAVE REPORTS И ОТЧЕТЫ В ПРИЛОЖЕНИЯХ DELPHI

В Delphi 7 генератор отчетов Rave Reports является основным средством создания отчетов и его компоненты устанавливаются в палитре компонентов по умолчанию на странице Rave.

12.1. ГЕНЕРАТОР ОТЧЕТОВ RAVE REPORTS

Генератор отчетов Rave Reports 5.0 разработан фирмой Nevrona и входит в состав Delphi 7 в качестве основного средства для создания отчетов. Он состоит из трех частей:

- ядро генератора отчетов обеспечивает управление отчетом и его предварительный просмотр, и отправку на печать. Исполняемый код ядра сервера включается в приложение Delphi, делая его полностью автономным при работе с отчетами на компьютере клиента;
- визуальная среда разработки отчетов Rave Reports предназначена для разработки самих отчетов. Она позволяет добавлять к отчету страницы, размешать на них графические и текстовые элементы управления, подключать к отчетам источники данных и т. д. Отчеты сохраняются в файлах с расширением rav и должны распространяться совместно с приложениями, использующими их;

• компоненты Rave Reports расположены на странице **Rave** Палитры компонентов Delphi. Они обеспечивают управление отчетами в приложении.

Генератор отчетов устанавливается при инсталляции Delphi в папку \Delphi7\Rave5. Исходные коды компонентов разработчикам в Delphi недоступны.

12.2. КОМПОНЕНТЫ RAVE REPORTS И ИХ НАЗНАЧЕНИЕ

Компоненты для создания отчетов и управления расположены на странице Rave палитры компонентов. Они делятся на следующие функциональные группы.

• компонент отчета TRvproject, с точки зрения приложения, и есть отчет. Он обеспечивает загрузку заранее созданного в визуальной среде Rave Reports отчета из файла с расширением rav.

• компонент управления отчетом TRvSystem обеспечивает работу приложения с отчетом. Взаимодействуя с компонентом отчета, с одной стороны, и сервером отчета Rave Reports, с другой, этот компонент обеспечивает просмотр и печать отчетов.

• компоненты соединения с источниками данных предназначены для подключения различных источников данных к отчетам. При этом могут использоваться технологии доступа к данным ADO, BDE, dbExpress.

К этой группе относятся компоненты:

- TRvCustomConnection;
- TRvDataSetConnection;
- TRvTableConnection;
- TRvQueryConnection.

• Компоненты преобразования данных позволяют конвертировать отчеты из формата данных Rave Reports в другие форматы (текстовый, PDF, HTML, RTF), а также распечатывать или просматривать отчеты.

К этой группе относятся компоненты:

- TRvNDRWriter;
- TRvRenderHTML;
- TRvRenderPreview;
- TRvRenderRTF;
- TRvRenderPrinter;
- TRvRenderText.
- TRvRenderPDF;

12.3. ОТЧЕТ В ПРИЛОЖЕНИИ DELPHI

Основой отчета является файл отчета с расширением «*.rav». Он создается в визуальной среде разработки Rave Reports и может содержать произвольное число страниц. Каждая страница может быть оформлена

графическими или текстовыми элементами или отображать данные из какой-либо базы данных. Другими словами, файл RAV — это проект будущего отчета, содержащий общую информацию об отчете, оформление его страниц и правила их заполнения.

После создания проект отчета необходимо связать с приложением Delphi. Для этого используется компонент TRvProject. Этот компонент обеспечивает представление отчета в приложении.

Но этого недостаточно, чтобы просмотреть или напечатать отчет. Для выполнения этих операций используется код ядра генератора отчета, который автоматически прикомпилируется к исполняемому коду приложения при переносе на любую форму проекта компонентов TRvProject и TRvSystem. Для управления операциями печати и просмотра в проекте должен присутствовать компонент TRvSystem.

Для того чтобы приложение Delphi могло выполнять функции печати отчетов, разработчик должен выполнить следующий набор операций.

1. При помощи визуальной среды разработки Rave Reports необходимо создать проект отчета и сохранить его.

2. Перенести в проект приложения в Delphi компонент TRvProject и связать его с файлом проекта отчета при помощи свойства ProjectFile.

3. Перенести в проект приложения в Delphi компонент TRvSystem и связать его с компонентом TRvProject. Для этого используется свойство Engine компонента TRvProject.

4. Написать код приложения, обеспечивающий просмотр и печать отчета (при необходимости и другие операции), используя методы компонента TRvProject.

12.4. КОМПОНЕНТ ОТЧЕТА TRVPROJECT

Компонент TRvProject обеспечивает представление в приложении отчета. Для того чтобы связать проект отчета Rave Reports с компонентом, используется свойство

property ProjectFile: string;

До начала печати необходимо связать компонент TRvProject с компонентом управления отчетом TRvSystem. Для этого достаточно передать в свойстве

property Engine: TRpComponent;

ссылку на компонент TRvSystem.

При необходимости вы можете загрузить отчет из внешнего файла или потока:

procedure LoadFromFile(FileName: String);
procedure LoadFromStream(Stream: TStream);

Загруженный отчет становится текущим.

Кроме этого существует и пара методов для сохранения отчета:

function SaveToFile(FileName: String);
procedure SaveToStream(Stream: TStream);

Файл проекта отчета можно включить в состав исполняемого файла приложения. Для этого используется свойство

property StoreRAV: Boolean;

Отправить отчет на печать можно методом

procedure Execute;

или же методом

procedure ExecuteReport(ReportName: string);

который позволяет направить на печать отчет, заданный параметром ReportName. Он должен соответствовать имени отчета, хранящемуся в свойстве ReportName компонента TRvProject.

Отчет, содержащийся в компоненте Trvproject, может быть открыт для редактирования методом

procedure Open;

Не открывая отчет, вы не сможете использовать большинство свойств и методов компонента. Дело в том, что при открытии компонент загружает отчет из файла проекта или прикомпилированного кода (в случае использования свойства StoreRAV).

Сохранение и закрытие отчета соответственно выполняются методами

procedure Save; procedure Close;

Кроме этого, действия, аналогичные методам open и close, выполняются свойством

property Active: Boolean;

Если свойству присвоить значение True — отчет открывается, иначе — закрывается.

До и после открытия и закрытия отчета вызывается четверка методов-обработчиков:

property BeforeOpen: TNotifyEvent;
property AfterOpen: TNotifyEvent;
property BeforeClose: TNotifyEvent;

property AfterClose: TNotifyEvent;

12.5. КОМПОНЕНТ УРАВЛЕНИЯ ОТЧЕТОМ

Компонент управления отчетом TRvSystem обеспечивает выполнение основных операций с отчетом из приложения. В приложении он должен быть связан с компонентом TRvProject. Этого вполне достаточно, чтобы компонент TRvSystem выполнил свою работу. У разработчика нет необходимости вызывать какие-либо методы компонента, чтобы направить отчет на печать.

12.6. ВИЗУАЛЬНАЯ СРЕДА СОЗДАНИЯ ОТЧЕТОВ

Визуальная среда создания отчетов входит в состав генератора отчетов Rave Reports 5.0. В отличие от генератора отчетов Quick Report, который поставлялся с Delphi 6 и более ранними версиями, визуальная среда в Rave Reports значительно облегчает самый трудоемкий этап в процессе создания отчета и его включения в состав приложения — постраничную разработку шаблона отчета.

Под шаблоном отчета мы подразумеваем совокупность страниц отчета с расположенными на них графическими и текстовыми элементами оформления, а также свойствами и правилами создания отчета, сохраненными в файле с расширением rav.

Кроме этого, средствами визуальной среды шаблон отчета может быть подключен к различным источникам данных. При этом могут использоваться следующие технологии доступа к данным:

- ADO;
- dbExpress;
- BDE.

Визуальная среда Rave Reports открывается из меню Tools | Rave Designer главного окна Delphi или при двойном щелчке на компоненте TRvProject.

12.7. ИНСТРУМЕНТАРИЙ ВИЗУАЛЬНОЙ СРЕДЫ СОЗДАНИЯ ОТЧЕТОВ

Пользовательский интерфейс визуальной среды создания отчетов Rave Reports во многом напоминает среду разработки Delphi. В верхней части окна располагается панель инструментов, состоящая из набора кнопок слева и палитры инструментов справа. В палитре инструментов располагаются не только элементы оформления отчетов, но и инструменты для их настройки и управления.

Рассмотрим подробнее назначение каждой закладки палитры инструментов. Первые четыре содержат элементы оформления отчетов:

- Drawing — графические элементы оформления;
- Bar Code — различные типы штрихкодов;
- Standard — элементы оформления, позволяющие размещать на страницах отчета текст и изображения;
- Report — элементы оформления, предназначенные для отображения данных из внешних источников данных, подключенных к отчету.

Остальные закладки содержат инструменты управления и настройки страниц и элементов оформления:

- Zoom — управляет увеличением текущей страницы;
- Colors — позволяет установить цвета элементов оформления и страниц;
- Lines — задает стиль и толщину линий элементов оформления;
- Fills — задает стиль заполнения элементов оформления;
- Fonts — позволяет задать параметры шрифта для текста;
- Alignment — управляет выравниванием элементов оформления на странице.

Закладка Page Designer содержит еще один блокнот, каждая из страниц которого соответствует одной странице отчета. Когда вы добавляете к отчету новую страницу, здесь появляется еще одна закладка с именем новой страницы. На страницы можно переносить элементы оформления, изменять их размеры и местоположение. На страницу также можно спроектировать измерительную сетку, которая поможет размещать и выравнивать элементы оформления. Обрамляют страницу вертикальная и горизонтальная линейки.

На страницу можно переносить элементы оформления из Палитры инструментов, и затем элементы оформления можно выделять, настраивать их свойства, перемещать и удалять.

Закладка Event Editor обеспечивает создание методов-обработчиков событий для отчетов, страниц, элементов оформления и т. д.

Правую часть окна среды разработки занимает панель проекта отчета. Дерево проекта содержит все его составные части. При двойном щелчке на элементе дерева он отображается на странице в центральной части.

В левой части окна среды разработки располагается аналог Инспектора объектов Delphi, в котором доступны свойства текущего элемента. В нижней части этой панели отображается подсказка для текущего свойства.

12.8. СТАНДАРТНЫЕ ЭЛЕМННТЫ ОФОРМЛЕНИЯ

Стандартные элементы формления используются так же, как и компоненты в Delphi. Выбранный элемент переносится из Палитры инструментов на страницу отчета. Здесь его можно разместить в нужном месте, изменить его размеры и настроить свойства. Свойства элемента оформления доступны на панели слева. Кроме этого, наиболее важные

визуальные свойства (цвет, стиль линий и заполнения и т. д.) вынесены на палитру инструментов.

После переноса на страницу имя элемента оформления также появляется в дереве проекта.

12.9. ЭЛЕМНТЫ ДЛЯ ПРЕДСТАВЛЕНИЯ ТЕКСТА И ИЗОБРАЖЕНИЙ

На странице Standard палитры инструментов расположены элементы оформления, предназначенные для отображения текста и изображений.

Для представления однострочного текста имеется простой элемент оформления Text. Текст задается свойством Text. Другие стандартные свойства позволяют настраивать шрифт, цвет и т. д. Кроме этого, свойство Rotation позволяет повернуть текст на любой угол в диапазоне от 0 до 360 градусов.

Для представления многострочного текста используется элемент оформления Memo. Текст задается свойством Memo.

Если вам необходимо объединить несколько элементов оформления в группу (например, несколько строк текста и изображений в заголовке страницы) и использовать их на странице совместно, применяется невизуальный элемент оформления Section. Размещенные на нем другие элементы как бы оказываются на самостоятельной странице, ограниченной элементом section. Вы можете выделять и перемещать отдельные элементы, но делать это только внутри секции. А при перемещении секции по странице все расположенные на ней элементы также перемещаются вместе с ней. Выравнивание элементов тоже работает по границам секции.

Для представления изображений используется элемент оформления Bitmap. Он позволяет оформлять отчеты изображениями, сохраненными в файлах в формате BMP. Для загрузки изображения используется диалог, открывающийся при щелчке на кнопке свойства image. Также при помощи свойства FileLink можно связать элемент с файлом изображения и при печати отчета оно будет загружено. Кроме этого, изображение можно загрузить из базы данных. Для этого используются свойства Dataview (определяет объект просмотра данных) и DataField (задает поле изображения).

Изображение можно масштабировать. Для этого используется свойство Matchside. При его значениях msHeight, msWidth, msBoth изображение соответственно масштабируется по горизонтали, вертикали или в двух измерениях. Естественно, при этом может произойти искажение изображения. А вот при значении msInside изображение будет масштабировано пропорционально.

Аналогичными свойствами обладает элемент оформления MetaFile. Но изображение в формате WMF загружается в него при помощи свойства FileLink или DataField.

Невизуальный элемент оформления FontMaster позволяет задать единые свойства шрифта для группы элементов оформления. Его свойство Font

определяет шрифт. И этот шрифт будет использоваться всеми элементами, в чьих свойствах FontMirror будет указан данный элемент FontMaster.

Невизуальный элемент оформления pageNuminit обеспечивает нумерацию страниц, начиная с той, на которой он расположен. Свойство initvalue задает номер, с которого начинается отсчет нумерации. А при помощи свойств initoataview и initoataFieid можно загрузить это значение из базы данных.

12.10. ГРАФИЧЕСКИЕ ЭЛЕМЕНТЫ УПРАВЛЕНИЯ

Графические элементы оформления расположены на странице **Drawing** Палитры инструментов.

Это основные три элемента оформления для рисования линий:

- HLine — горизонтальная линия;
- VLine — вертикальная линия;
- Line — универсальная линия.

Все эти элементы имеют идентичный набор свойств, позволяющих задавать параметры и размер линий. Кроме этого, элемент Line позволяет развернуть линию на любой угол. Это можно сделать при помощи мыши.

Элементы оформления square и Rectangle изображают квадрат и прямоугольник соответственно.

Элементы Ellipse и circle изображают эллипс и круг.

12.11. ШТРИХКОДЫ

На странице Bar Code разработчику доступны шесть элементов оформления, позволяющие включать в отчеты штрихкоды. Все они реализуют различные стандарты, но значение для кодирования у всех задается одним свойством Text. Элементы PostNetBarCode, I2of5BarCode, UPCBarCode и EANBarCode позволяют вводить только числа, а элементы Code39BarCode и Godei28BarCode могут работать и с буквенно-цифровыми последовательностями.

1. Код PostNet используется почтовой службой США, содержит код адреса.
2. Перемежающийся код I2of5 служит для представления числовых последовательностей. Перемежающимся назван потому, что цифры в последовательности попеременно кодируются штрихами и пробелами.
3. Код Code39 предназначен для кодирования цифр, заглавных букв латинского алфавита и некоторых других символов. Для представления символа используются пять штрихов и четыре пробела.
4. Код Codel28 позволяет хранить первые 128 символов ASCII.
5. Код UPC (Universal Product Code) может содержать только цифры. Разработан для маркировки продуктов. Код может включать 12 цифр.

6. Код EAN (European Article Numbering system) подобен UPC. Код может включать 13 цифр. Первые две отводятся под код страны-производителя.

Для всех элементов значение для кодирования можно загрузить из базы данных при помощи свойства FileLink или DataField.

При необходимости можно рассчитать и напечатать контрольную сумму. Свойство usechecksum при значении True рассчитывает ее, а свойство Printchecksum, будучи установленным в значение True, печатает.

Штрихкод можно развернуть, но только с дискретностью 90°. Для этого используется свойство BarCodeRotation.

12.12. ВНЕШНИЕ ИСТОЧНИКИ ДАННЫХ

Все объекты, обеспечивающие доступ к внешним источникам данных из отчетов проекта, собраны в словаре просмотра данных Data View Dictionary. Новый объект создается командой File | New Data Object главного меню.

12.13. СОЕДИНЕНИЕ С ИСТОЧНИКОМ ДАННЫХ

В первую очередь необходимо создать соединение с источником данных. Для этого в диалоге (рис. 2) необходимо выбрать объект соединения **Database Connection** и, после нажатия кнопки **Next**, в следующем окне выбрать одну из трех возможных технологий доступа к данным: ADO, dbExpress, BDE. В зависимости от сделанного выбора настраиваются параметры соединения.

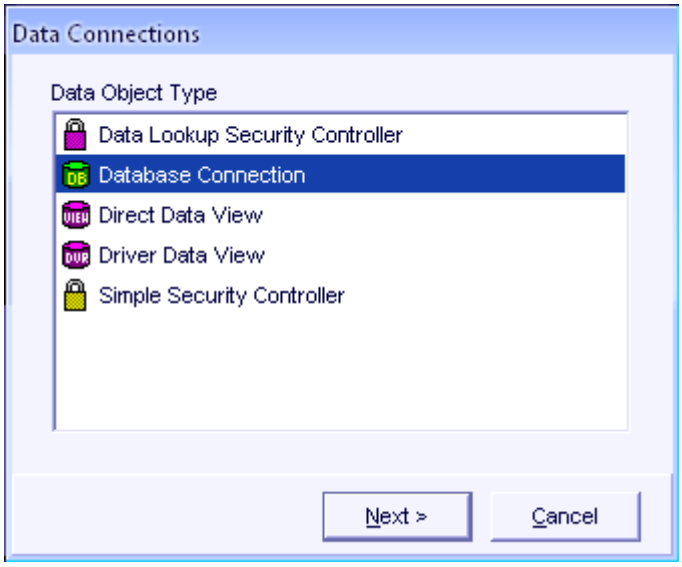

Рис. 2. Окно Data Connections

После завершения настройки готовое соединение появляется в списке Data View Dictionary.

На втором этапе к работающему соединению можно подключать объекты просмотра данных. В диалоге Data Connections необходимо выбрать тип объекта доступа к данным Driver Data View и нажать на кнопку Next. Затем в следующем окне нужно выбрать одно из существующих соединений. После этого появляется диалог Query Advanced Designer (рис.3).

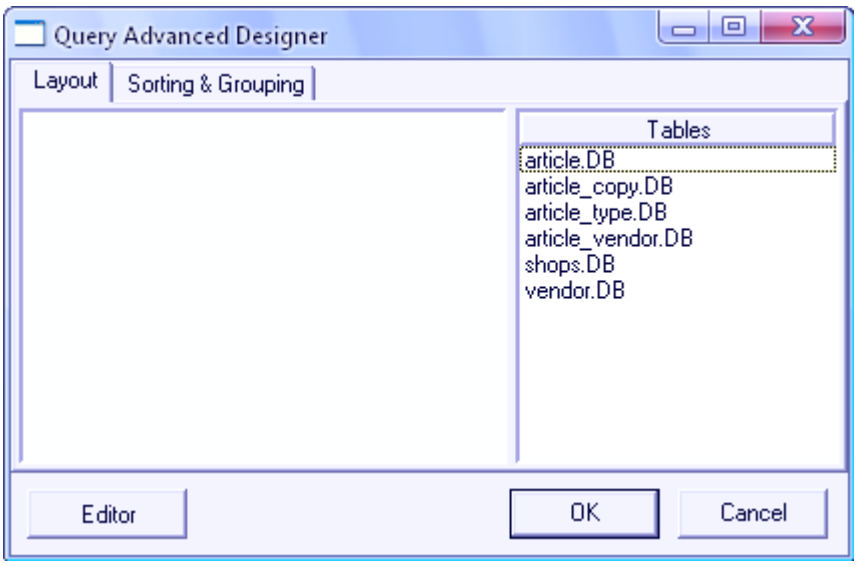

Рис. 3. Окно Query Advanced Designer

В этом диалоговом окне нужно выбрать из списка доступных таблиц источника данных необходимые и задать требуемые соотношения между полями. Страница Layout (рис. 4) двухстраничного диалога позволяет выбирать таблицы. Нужная таблица выбирается из списка Tables в правой части и перетаскивается в левую часть, где каждая таблица представляется в виде списка полей. Необходимые поля также можно выбирать для будущего использования. Перетаскиванием полей между таблицами можно задавать внешние ключи.

Страница Sorting & Grouping позволяет выбрать поля для сортировки и группировки. Здесь в списке слева представлены выбранные ранее таблицы и поля. Поля можно переносить в список Sort Fields справа. Поля в этом списке будут использованы для сортировки наборов данных. А из полей для сортировки можно выбрать поле для группировки. Для этого используется список Group By.

При нажатии на кнопку Editor появляется редактор с текстом запроса SQL, реализующий все сделанные ранее настройки. При необходимости его можно исправить вручную.

После завершения настройки просмотра новый объект просмотра появляется в словаре просмотра данных Data View Dictionary и может использоваться в отчетах.

Созданный запрос просмотра доступен через его свойство Query.

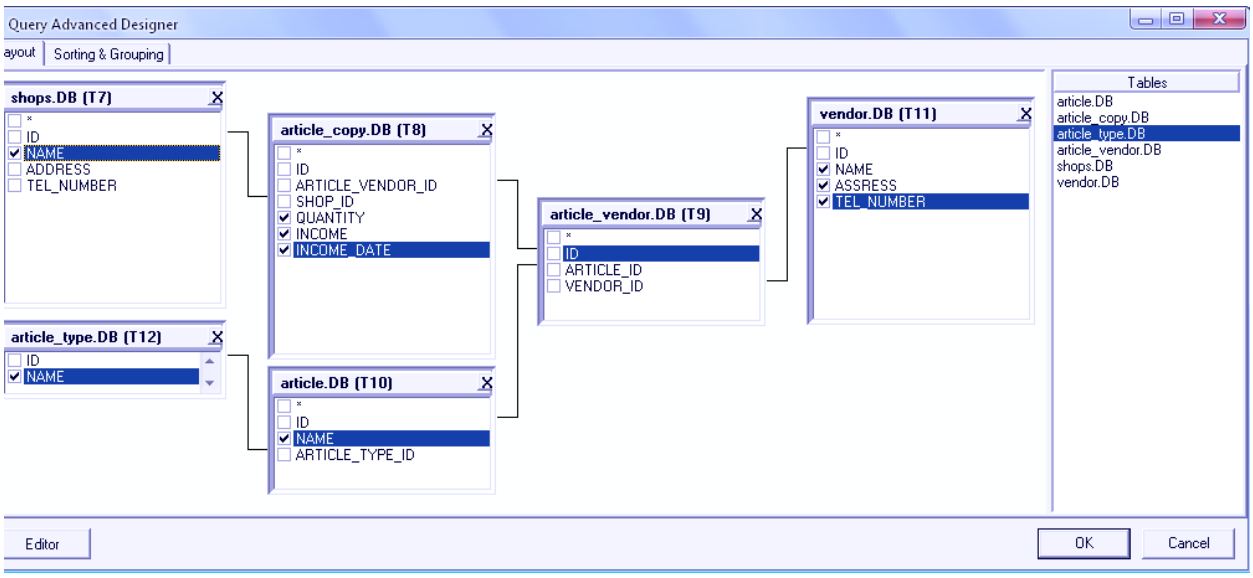

Рис. 4. Окно Query Advanced Designer вкладка Layout

12.14. ОТОБРАЖЕНИЕ ДАННЫХ В ОТЧЕТАХ

Для представления данных в отчетах предназначены специализированные элементы оформления, представленные на странице Report палитры инструментов.

Они делятся на две функциональные группы:

в первую группу выделены элементы, обеспечивающие размножение строк в отчете, их заполнение данными, а также группировку при создании сложных отчетов. Эти элементы мы будем называть структурными.

во второй группе объединены элементы оформления, созданные на основе стандартных и обеспечивающие отображения текущего значения конкретного поля таблицы просмотра.

12.15. СТРУКТУРНЫЕ ЭЛЕМЕНТЫ ОТЧЕТА

Основой отчета, использующего просмотры баз данных, является элемент Region. Он создает в отчете область, предназначенную для размещения любых других элементов и определяющую часть страницы отчета, отведенную под отображение данных.

При создании отчета, использующего базу данных, этот элемент переносится на страницу в первую очередь. Затем добавляются элементы Band и DataBand.

Элемент Band создает полосу, на которой можно располагать стандартные элементы оформления. Он служит для оформления заголовков, сносок, врезок и других статичных фрагментов оформления отчетов, которые не изменяются при печати просмотра данных.

Элемент DataBand создает полосу, моделирующую строку просмотра данных. На ней располагаются элементы отображения данных, которые будут рассмотрены ниже. При печати отчета для каждой строки печатается новый экземпляр полосы элемента DataBand со всеми расположенными на ней элементами оформления. Таким образом и получается отчет, отображающий строка за строкой весь набор данных.

Важнейшее свойство Bandstyle определяет роль и поведение полосы в отчете. С ним связано диалоговое окно Band Style Editor (рис. 5), которое отображает взаимосвязь полос в области Region отчета и позволяет задать поведение текущей полосы.

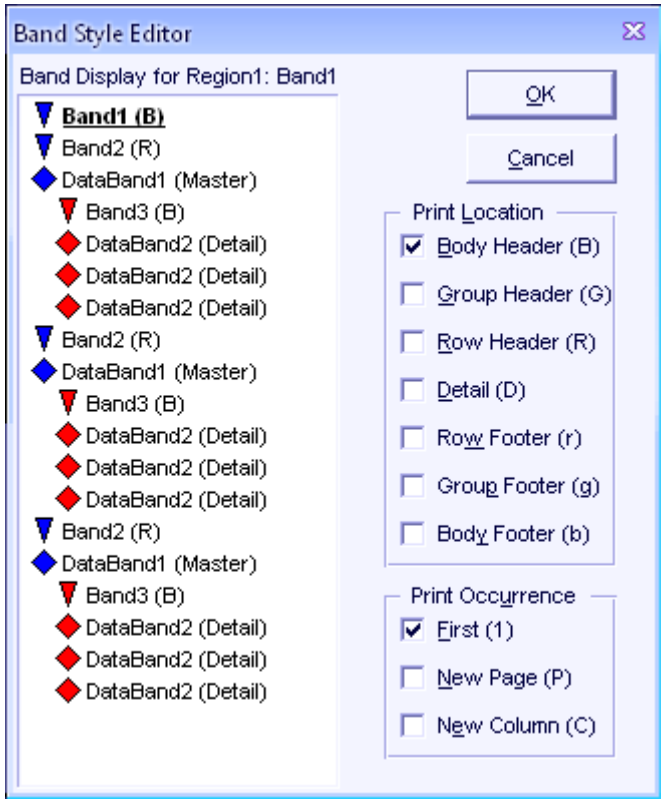

Рис. 5. Окно Band Style Editor

В левой части диалога отображается список всех полос отчета с их взаимосвязями (отношениями «один – ко – многим», группировкой, вложенностью и т. д.), текущая полоса выделяется жирным шрифтом с подчеркиванием. Имя каждой полосы отображается трижды. И это не ошибка разработчиков, а желание показать, что каждая полоса размножается для печати записей просмотра данных.

Группа флажков Print Location в правой части диалогового окна определяет назначение полосы. А группа Print Occurrence задает, в каком месте отчета появляется полоса:

1. Body Header (B) — заголовок отчета, печатается в начале отчета;
2. Group Header (G) — заголовок группы, печатается в начале группы записей, объединенных в просмотре данных выражением GROUP BY;
3. Row Header (R) — заголовок записи, печатается в начале каждой записи просмотра данных;
4. Detail (D) — печатается в начале подчиненного набора записей, входящего в отношение «один – ко – многим»;
5. Row Footer (r) — окончание строки, печатается в конце каждой записи просмотра данных;
6. Group Footer (g) — окончание группы, печатается в конце группы записей, объединенных в просмотре данных выражением GROUP BY;
7. Body Footer (r) — окончание отчета, печатается в конце отчета;
8. First (1) — печатается один раз в начале отчета (титул отчета);

9. New Page (P) — печатается в начале каждой страницы отчета;
10. New Column (C) — печатается в начале каждой колонки отчета.

12.16. ЭЛЕМЕНТЫ ОТОБРАЖЕНИЯ ДАННЫХ

Элементы отображения данных представляют собой модифицированные стандартные элементы, размещаются на структурных элементах отчета и отображают данные из связанных с ними полей просмотра данных. Они расположены на странице Report Палитры инструментов.

Элемент DataText предназначен для представления строковых или числовых значений полей связанного просмотра данных.

Элемент DataMemo используется при необходимости показать данные в формате Memo или BLOB.

Элемент calcText обеспечивает выполнение одной из агрегатных функций над значениями связанного поля и представление результата. Тип операции выбирается в свойстве calcType.

Невизуальный элемент DataMirrorSection, так же как и его предок Section, объединяет группу других элементов для совместного использования.

Кроме перечисленных элементов, еще один элемент способен отображать данные из поля просмотра. Это стандартный элемент оформления Bitmap со страницы Standard Палитры инструментов

Все перечисленные элементы (в том числе и элемент Bitmap) связываются с просмотром данных и полем одинаково.

Свойство Dataview определяет, какой просмотр данных используется элементом.

Свойство DataField задает поле просмотра, значения которого будут отображаться элементом.

12.17. ПРОСМОТР И ПЕЧАТЬ ОТЧЕТА

За выполнение любых операций с ним отвечает компонент TRvSystem.

При стандартной настройке этого компонента, при печати или предварительном просмотре отчета всегда отображается диалог настройки печати (рис. 6). Если отображение этого диалога необходимо, печать текущего отчета компонента TRvproject, с которым связан данный компонент TrvSystem, осуществляется методом *procedure Execute;* любого из этих компонентов.

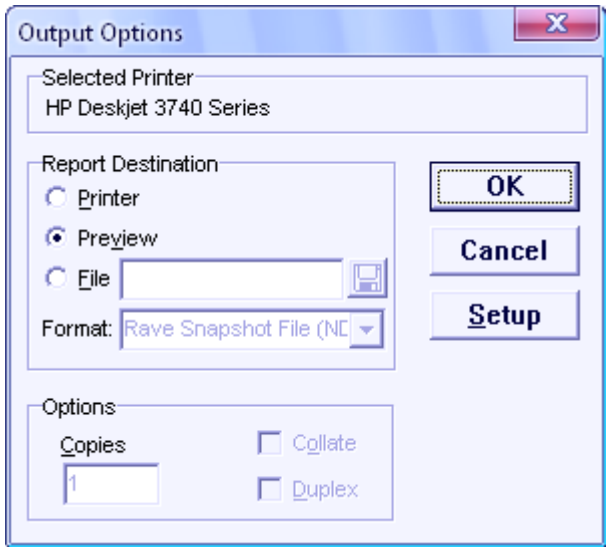

Рис. 6. Диалог настройки печати

Примечание: *для работы с отчетом, использующим внешний источник данных, необходимо в среде Delphi добавить компонент RvDataSetConnection, а затем, в визуальной среде разработки Rave, создать компонент Direct Data View. Пример работы с отчетом БД продемонстрирован на компакт диске в каталоге DBReport.*

Задание: на основе данных предыдущих лабораторных работ создать запрос на основе нескольких таблиц с применением группировки, условия и агрегирующих функций. В визуальной среде Rave Reports создать проект отчета, на основе БД. Включить проект отчета в приложение БД.

Контрольные вопросы:

1. Какие типы данных, определенные стандартом ANSI вы знаете?
2. Как устранить избыточность выводимых данных?
3. Что такое квалификационный выбор?
4. Принцип работы оператора LIKE?
5. Как объединить данные нескольких таблиц с помощью одной команды, используя справочную целостность?
6. Зачем нужно свойство Band Style в полосе Band / DataBand?
7. Без использования каких компонентов, невозможна работа с отчетом, который использует внешние данные?

Приложение

ТИПЫ ДАННЫХ ANSI

ТИП	СИНОНИМ	ОПИСАНИЕ
TEXT (ТЕКСТ)		
CHAR	CHARACTER	Строка текста в формате, зависящим от реализации. Размер аргумента равен максимальной длине строки. Значения этого типа должны быть заключены в одиночные кавычки, например 'text'. Две рядом стоящие одиночные кавычки ('') внутри строки будет пониматься как одна одиночная кавычка (').
EXACT NUMERIC (ТОЧНОЕ ЦЕЛОЕ)		
DEC	DECIMAL	Десятичное число, то есть число, которое может иметь десятичную точку. Здесь аргумент размера имеет две части: точность и масштаб. Масштаб не может превышать точность. Сначала указывается точность, разделительная запятая и далее аргумент масштаба. Точность указывает, сколько значащих цифр имеет число. Максимум определяется реализацией. Масштаб указывает максимальное число цифр справа от десятичной точки. При масштабе равном нулю, поле эквивалентно целому числу.
NUMERIC		Такое же, как и DECIMAL, за исключением того, что максимальное десятичное не может превышать аргумента точности.
INT	INTEGER	Число без десятичной точки. Эквивалентно DECIMAL, но без цифр справа от десятичной точки, то есть с масштабом равным нулю. Аргумент размера не используется (он автоматически определяется реализацией).
SMALLINT		Такое же, как INTEGER, за исключением того, что, в зависимости от реализации, размер по умолчанию может (или не может) быть меньшее, чем INTEGER.
APPROXIMATE NUMERIC (ПРИБЛИЗИТЕЛЬНОЕ ЧИСЛО)		
FLOAT		Число с плавающей запятой на основе 10 показательной функции. Аргумент размера состоит из одного числа определяющего минимальную точность.
REAL		Такое же, как FLOAT, за исключением того, что никакого аргумента размера не используется. Точность устанавливается реализацией.
DOUBLE PRECISION		Такое же, как REAL, за исключением того, что реализационно - определяемая точность для DOUBLE PRECISION должна превышать реализационно - определяемую точность REAL.