МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ
УЧРЕЖДЕНИЕ ВЫСШЕГО ПРОФЕССИОНАЛЬНОГО ОБРАЗОВАНИЯ
«САМАРСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»

Кафедра безопасности информационных систем

А. Н. Крутов

БАЗЫ ДАННЫХ

Утверждено редакционно-издательским советом университета
в качестве лабораторного практикума

Самара
Издательство «Самарский университет»
2013

УДК 681.3.07
ББК 32.973.26
К 84

Рецензенты: канд. техн. наук А. В. Костров,
канд. техн. наук С. Е. Агафонова

Крутов, А. Н.

К 84 Базы данных : лабораторный практикум / А. Н. Крутов. Самара : Изд-во «Самарский университет», 2013. – 52 с.

В практикуме приводятся достаточные для выполнения лабораторных работ теоретические сведения. Подробно описывается ход выполнения лабораторных работ с использованием программ Power Architect 0.9.7, СУБД Firebird 2.0, и IB Expert 2.0.

Практикум предназначен для студентов специальностей 090102.65 Компьютерная безопасность и 090103.65 Организация и технология защиты информации. Материалы учебного пособия позволяют выполнить комплекс лабораторных работ по дисциплине «Системы управления базами данных».

УДК 681.3.07
ББК 32.973.26

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Хотим мы этого или не хотим, но базы данных стали неотъемлемой частью нашей повседневной жизни. За примерами использования баз данных не надо далеко ходить. Достаточно вспомнить, как мы расплачиваемся за купленные товары в супермаркете или работаем с электронным каталогом библиотеки. Подготовка квалифицированных специалистов требует развития у них практических навыков проектирования и разработки баз данных.

В лабораторном практикуме основное внимание уделено вопросам проектирования и реализации баз данных. В нем не рассматриваются вопросы построения приложений с базами данных, т.к. таким образом весь процесс обучения был бы ориентирован на какую-то одну среду разработки или одну технологию программирования.

Задача учебного пособия – способствовать развитию у студентов творческих способностей путем освоения процесса проектирования базы данных и ее реализации.

Для всего лабораторного практикума достаточно наличие всего трех программных продуктов, причем все они являются бесплатными.

Так, для проектирования баз данных используется программа Power Architect 0.9.7, которую бесплатно можно скачать со страницы http://www.sqlpower.ca/page/architect. В качестве СУБД используется Firebird 2.0, которую также можно бесплатно скачать с сайта http://www.-firebirdsql.org. Для удобной работы с данной СУБД используется программа IB Expert 2.0, которую можно найти по адресу: http://ibexpert.net/ibe. Данная программа является бесплатной для граждан бывшего СССР.

Выбор именно данных программ для основы курса лабораторных работ обуславливается желанием сделать курс как можно более общим. Навыки, приобретенные студентами на примере СУБД Firebird могут быть с тем же успехом и с минимальными изменениями быть применены и к любой другой современной реляционной СУБД. Процесс же разработки логической модели базы данных не зависит от СУБД в принципе. Все бесплатные программы, используемые в курсе обучения по основным функциям не уступают платным аналогам.

В отличие от традиционной методики обучения, когда студентам даются какие-то предопределенные, четко поставленные задания, на примере которых проверяется, насколько тот или иной студент изучил материал, в настоящем курсе лабораторных работ составитель избрал совершенно другой путь.

По большому счету заданий по курсу СУБД всего два. Первое из них делается за первую лабораторную работу. Второе же делается все оставшееся время. Цель первого задания – на примере изучить основные функции программы Power Architect, с которой студенты будут работать в пер-

вых лабораторных работах. Второе же задание ставится достаточно кратко и степень его проработки целиком зависит от творческих способностей и умений конкретного студента. Оно включает в себя последовательный процесс разработки логической схемы базы данных, ее анализ с помощью правил нормализации, построение физической схемы базы данных, заполнение тестовыми данными, написание различных SQL – запросов и защита базы данных различными способами. Для удобства изложения второе задание разбито на отдельные лабораторные задания (2, 3, и т.д.), причем каждая следующая лабораторная работа является логическим продолжением предыдущей.

1. Лабораторная работа 1. Элементы модели «сущность-связь»

1.1. Цель работы

Изучить на простом примере элементы модели сущность-связь, научиться пользоваться программой Power Architect для проектирования баз данных, научиться исключать из схемы избыточные данные.

Время выполнения: 2 часа

1.2. Исходные данные

Каждый студент получает индивидуальное задание, которое представляет собой отчетную форму, которую выдает некая программа (см. приложение А).

1.3. Краткие теоретические сведения

Ключевыми элементами модели «сущность-связь» являются сущности, атрибуты, идентификаторы и связи.

Сущность – это некоторый объект, идентифицируемый в рабочей среде пользователя, нечто такое, за чем пользователь хотел бы наблюдать. Сущности одного и того же типа группируются в классы сущностей.

Атрибуты – это свойства, которые описывают характеристики сущности. В модели «сущность-связь» предполагается, что все экземпляры одного класса сущностей имеют одинаковые атрибуты.

В модели «сущность-связь» атрибуты бывают обычными, композитными и многозначными. В качестве примера композитного атрибута можно привести атрибут «Адрес», состоящий из группы атрибутов {Индекс, Населенный пункт, Улица}. Примером многозначного атрибута может служить атрибут «Телефон клиента» сущности «Клиент», который может содержать сразу несколько номеров телефонов данного клиента. В большинстве реализаций модели «сущность-связь» требуется, чтобы многозначные атрибуты преобразовывались в сущности.

Идентификаторы – это атрибуты экземпляров сущностей, с помощью которых эти экземпляры именуются или идентифицируются.

Например, экземпляры сущности класса «Студент» могут идентифицироваться по атрибуту «Номер зачетной книжки». Идентификатор экземпляра сущности состоит из одного или более атрибутов сущности. Идентификаторы, состоящие из нескольких атрибутов, называются **композитными идентификаторами**.

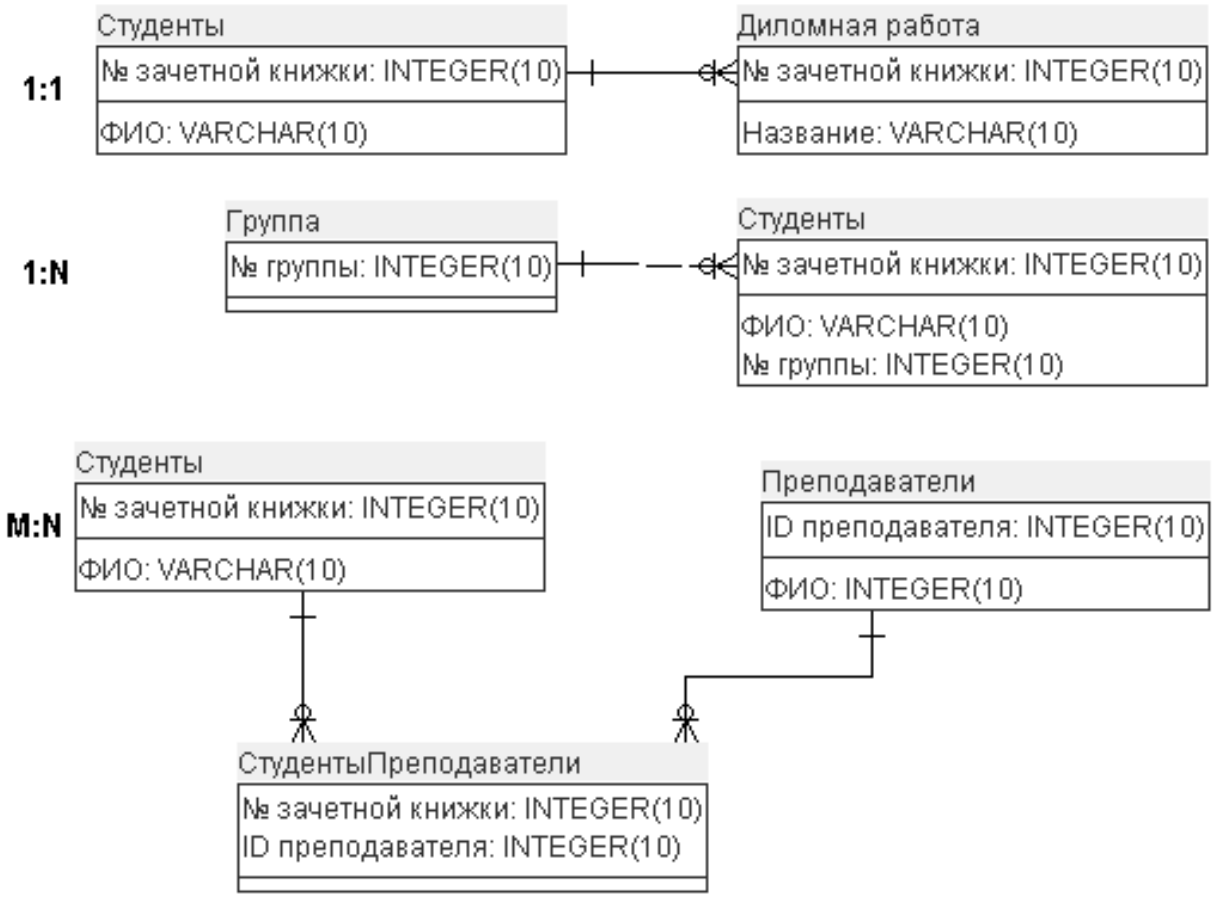
Взаимоотношения сущностей выражаются **связями**. Модель «сущность-связь» включает в себя **классы связей** и **экземпляры связей**.

Классы связей – это взаимоотношения между классами сущностей

Экземпляры связи – взаимоотношения между экземплярами сущностей.

Класс связей может затрагивать несколько классов сущностей. Число классов сущностей, участвующих в связи, называется **степенью связи**. Связи степени 2 весьма распространены, часто их еще называют **бинарными** связями.

Традиционно выделяют три типа бинарных связей: «один к одному», «один ко многим» и «многие ко многим» в зависимости от того, как экземпляры одного класса сущностей связаны с экземплярами другого класса сущностей:



1.4. Задание

Восстановить схему базы данных, по информации которой можно сгенерировать отчетную форму индивидуального задания.

1.5. Порядок выполнения работы

Пусть имеется следующая отчетная форма:

Результаты заплывов спортсменов, плывших по дорожке _____ на дистанции _____ [м] в текущем месяце

ФИО спортсмена	**Дата**	**Разряд**	**Результат (мин.сек.)**	**Отклонение от предварительного результата**
До 64 символов		До 20 символов	ЦЦЦ.ЦЦ	

Лучший результат ____ сек.
У пловца _____ тренер _____
Количество спортсменов, улучшивших результат

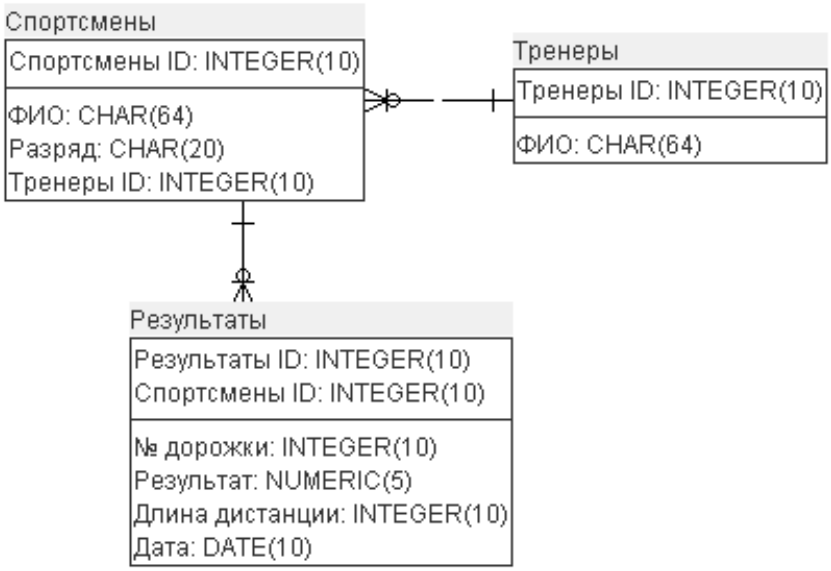
Анализируя приведенную выше отчетную форму, можно сделать заключение, что для хранения информации в базе данных целесообразно будет сделать три таблицы: спортсмены, тренеры и результаты, причем, таблица спортсменов будет ссылаться на тренеров, а результаты на спортсменов. Для создания схемы базы данных необходимо запустить программу Power Architect. Для добавления таблиц предназначена кнопка  . Добавим данные таблицы на диаграмму.

Анализируя отчетную форму можно сделать вывод, что не вся информация из нее должна быть непосредственно помещена в базу данных. Так, отклонение от предварительного результата нет смысла хранить в базе данных, если в ней будут содержаться все результаты. То же самое относится и к информации по лучшему результату и количеству спортсменов, улучшивших свой результат.

Для добавления атрибутов необходимо напротив соответствующей таблицы нажать правую кнопку мыши и выбрать **New Column**. В результате на экране появится форма, в которой вводится название атрибута, указывается его тип и вводятся его дополнительные характеристики.

Для того, чтобы связать таблицы между собой, необходимо, чтобы в таблицах были первичные ключи. В данном случае можно воспользоваться технологией создания суррогатных ключей. Для создания связей предназначены кнопки  и  . Первая из них предназначена для связи относительно независимых между собой таблиц, а вторая – для создания слабых сущностей, т.е. когда экземпляр одной сущности не имеет смысла без экземпляра другой сущности. Первым типом связей мы соединим таблицы

спортсменов с тренерами, а вторым – результаты со спортсменами. В результате получим окончательный результат в виде:



1.6. Контрольные вопросы

1. Перечислите основные элементы модели «сущность-связь».
2. Объясните, почему связь «многие ко многим» реализуется с помощью трех таблиц.
3. Опишите принцип создания естественных и суррогатных первичных ключей. Приведите достоинства и недостатки этих двух подходов.

2. Лабораторная работа 2. Построение логической схемы базы данных

2.1. Цель работы

Научиться разрабатывать модели сущность-связь, уметь выбирать нужные классы сущностей и связи между ними. Уметь выбирать требуемый уровень детализации задачи.

Время выполнения: 6 часов

2.2. Исходные данные

Каждый студент получает индивидуальное задание, которое не содержит четкой формулировки и определяет лишь направление дальнейшей работы (см. приложение В).

2.3. Краткие теоретические сведения

При проектировании информационной системы необходимо провести анализ целей этой системы и выявить требования к ней отдельных пользователей (сотрудников организации). Сбор данных начинается с изучения сущностей организации и процессов, использующих эти сущности. Сущности группируются по «сходству» (частоте их использования для выполнения тех или иных действий) и по количеству ассоциативных связей между ними (самолет – пассажир, преподаватель – дисциплина, студент – сессия и т.д.). Сущности или группы сущностей, обладающие наибольшим сходством и (или) с наибольшей частотой ассоциативных связей объединяются в предметные БД. (Нередко сущности объединяются в предметные БД без использования формальных методик – по «здравому смыслу».)

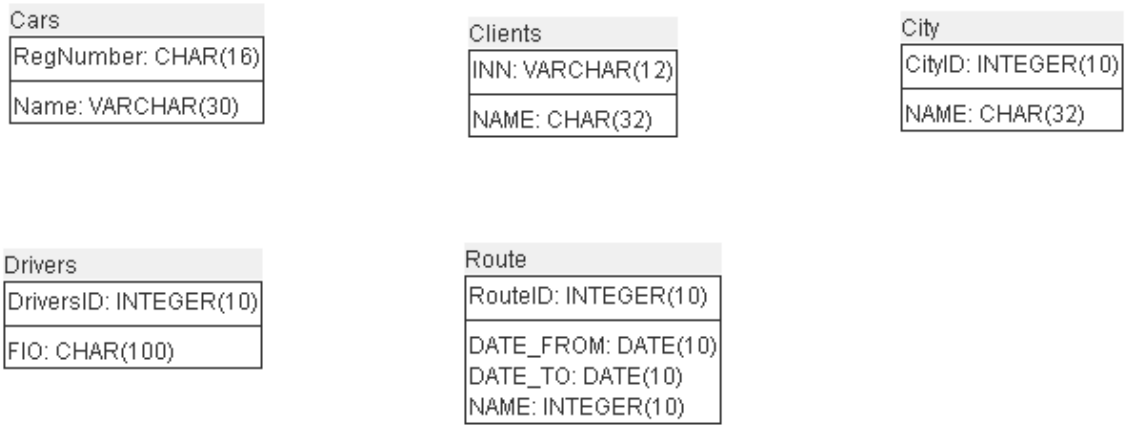
Основная цель проектирования БД – это сокращение избыточности хранимых данных, а следовательно, экономия объема используемой памяти, уменьшение затрат на многократные операции обновления избыточных копий и устранение возможности возникновения противоречий из-за хранения в разных местах сведений об одном и том же объекте.

2.4. Задание

Конкретизировать индивидуальное задание, определив необходимый уровень детализации модели. Построить диаграмму сущность-связь, соответствующую выбранному уровню абстракции.

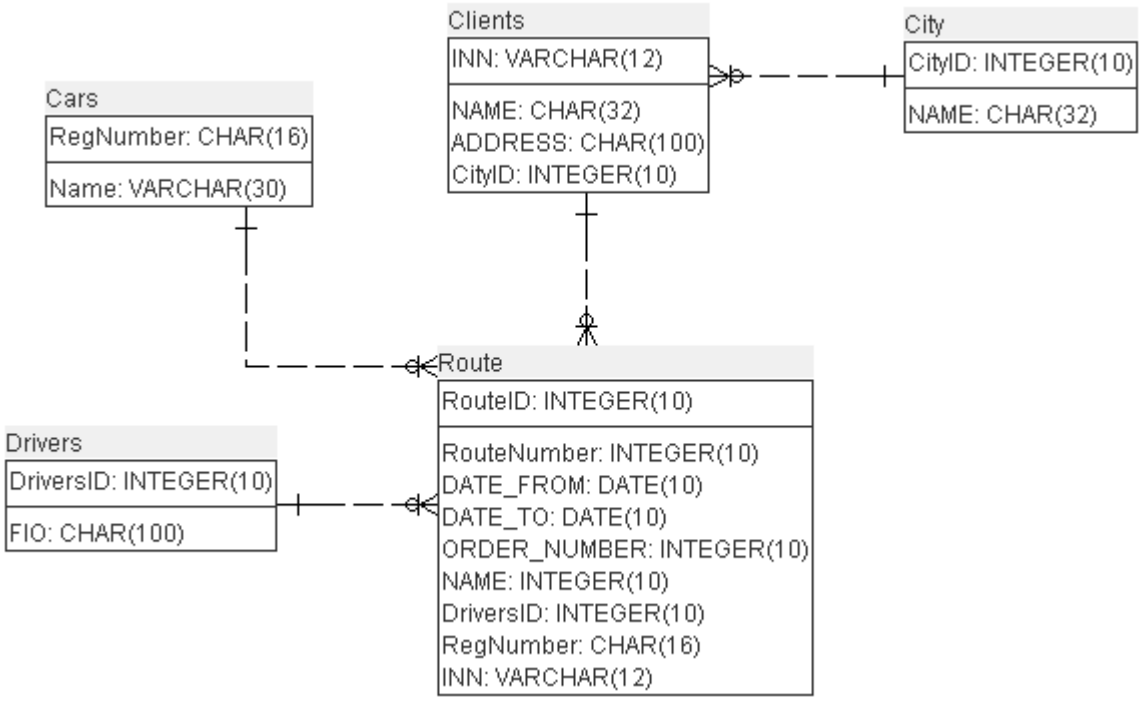
2.5. Порядок выполнения работы

Пусть требуется разработать структуру базы данных для системы учета транспортных перевозок. Предположим, что у нас есть парк автомобилей, штат водителей и клиенты, живущие в разных городах. Анализируя все вышеперечисленное, создаем сущности, описывающие предметную область задачи. Для краткости изложения не будем производить глубокую детализацию задачи.



Во избежание проблем при последующей конвертации, будем пользоваться исключительно английской раскладкой клавиатуры.

Предположим, что различные водители могут ездить на разных автомобилях, а за один маршрут можно посетить сразу несколько клиентов. Исходя из этого, организуем связь между таблицами следующим образом:



Примечание

Приведенный выше пример проектирования носит очень поверхностный характер. В реальности процесс проектирования является итерационным, т.к. в сложных системах невозможно сразу выделить все сущности предметной области. При детальной проработке того или иного бизнес-процесса появляются все новые и новые сущности и, соответственно, все новые и новые связи между ними.

2.6. Контрольные вопросы

1. Опишите последовательность процесса проектирования баз данных
2. Чем логическое проектирование отличается от физического?
3. На каком этапе проектирования баз данных происходит выбор целевой СУБД?

3. Лабораторная работа 3. Нормализация

3.1. Цель работы

Изучить сущность нормализации, и различные виды нормальных форм. Научиться проверять схему базы на удовлетворение определений нормальных форм и при необходимости уметь внести в нее соответствующие изменения.

Время выполнения: 2 часа

3.2. Исходные данные

Исходными данными являются результаты предыдущей лабораторной работы.

3.3. Краткие теоретические сведения

Нормализация – это процесс преобразования отношения, имеющего некоторые недостатки, в отношение, которое этих недостатков не имеет. Нормализацию можно использовать как критерий для определения желательности и правильности отношений.

Не все отношения одинаковы: некоторые из них более предпочтительны, чем другие. Таблица, отвечающая минимальному определению отношения, может иметь неэффективную или неподходящую структуру. Для некоторых отношений какое-либо изменение данных может привести к нежелательным последствиям, называемыми аномалиями. В определении нормализации недостатками являются именно аномалии.

Первая нормальная форма (1НФ)

Для того чтобы таблица находилась в первой нормальной форме, необходимо, чтобы выполнялись следующие условия:

1. Ячейки таблицы должны содержать одиночные значения и в качестве значений не допускаются ни повторяющиеся группы, ни массивы.
2. Все записи в одном столбце (атрибуте) должны иметь один и тот же тип.
3. Каждый столбец должен иметь уникальное имя, порядок следования столбцов в таблице несуществен.
4. В таблице не может быть двух одинаковых строк, порядок следования строк несуществен.

Для того, чтобы в таблице никогда не появилось двух одинаковых строк, достаточно, чтобы в таблице был уникальный идентификатор.

Вторая нормальная форма (2НФ)

Определение. Говорят, что если по значению атрибута A в таблице можно однозначно определить значение атрибута B, то атрибут A является **детерминантом** для атрибута B. Данная зависимость обозначается как A⇒B.

Отношение находится во второй нормальной форме, если все его неключевые атрибуты зависят от всего ключа целиком.

Третья нормальная форма (3НФ)

Определение. Если для атрибутов A, B и C некоторого отношения существуют зависимости A⇒B и B⇒C, то говорят, что атрибут C **транзитивно зависит** от атрибута A через атрибут B (при условии, что атрибут A функционально не зависит от атрибута B, ни от атрибута C)

Отношение находится в третьей нормальной форме, если оно находится во второй нормальной форме и не имеет транзитивных зависимостей.

Нормальная форма Бойса-Кодда (НФБК)

Отношение находится в НФБК, если каждый детерминант является ключом-кандидатом.

Четвертая нормальная форма (4НФ)

Определение. Многозначной зависимостью между атрибутами A, B и C называется случай, когда для каждого значения атрибута A имеется набор значений атрибута B и набор значений атрибута C. Однако значения атрибутов B и C не зависят друг от друга

Отношение находится в четвертой нормальной форме, если оно находится в НФБК и не имеет многозначных зависимостей.

3.4. Задание

Провести анализ созданной в предыдущей лабораторной работе схемы базы данных на нормальные формы (I-IV НФ). При необходимости внести изменения в схему, доведя ее, как минимум до III НФ.

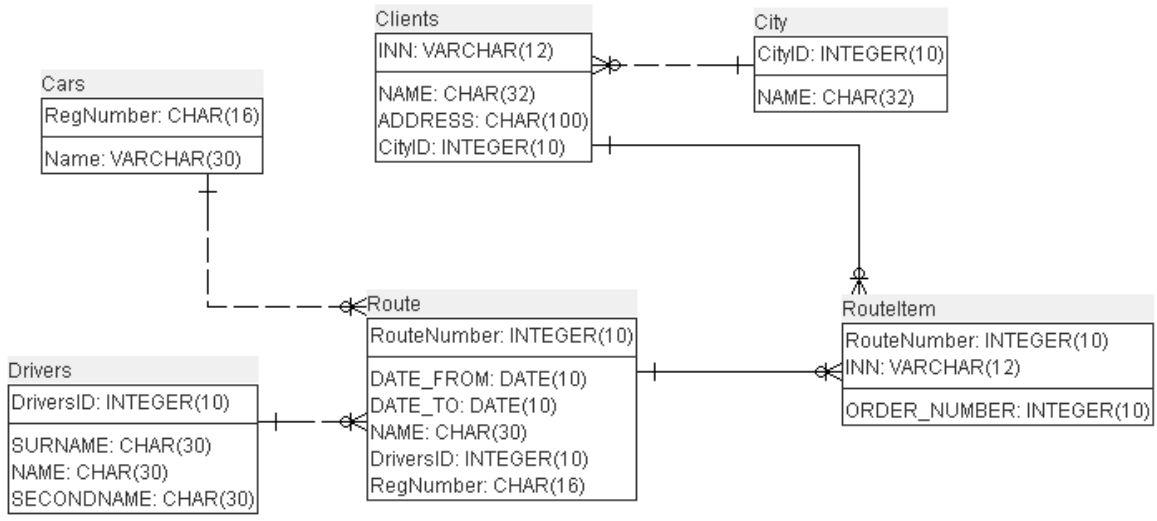
3.5. Порядок выполнения работы

Для того чтобы отношение находилось в первой нормальной форме необходимо, чтобы значения в атрибутах были атомарными. Для данной

задачи сомнения вызывают атрибуты ФИО в таблице водителей и поле адрес в таблице клиентов. Пусть для нас поле адрес будет неделимым, т.е. нигде в разрабатываемой системе не потребуется узнать отдельно улицу нахождения клиента или дом. Поле ФИО декомпозируем на три отдельных поля, т.к. впоследствии по этим атрибутам возможен поиск. Требование того, чтобы в таблице не было двух одинаковых строк выполняется за счет того, что в таблицах есть первичные ключи, которые должны быть уникальными. Остальные требования первой нормальной формы выполняются автоматически.

Вторая нормальная форма выполняется автоматически, т.к. все первичные ключи являются одиночными атрибутами.

Для проверки на третью нормальную форму достаточно проверить таблицы клиентов и маршрутов, т.к. в оставшихся таблицах всего по два атрибута. Очевидно, что в таблице клиентов все неключевые атрибуты друг от друга не зависят. Так, название никак не зависит от адреса и от города. Точно также адрес не зависит ни от названия ни от города, и город не зависит ни от названия ни от адреса. Что же касается таблицы маршрутов, то здесь очевидно наличие транзитивной зависимости. Так, поле ROUTENUMBER (номер маршрута) зависит от первичного ключа, а поле ORDERNUMBER (порядковый номер объезда) зависит от номера маршрута. Для того, чтобы избавиться от этой аномалии необходимо таблицу маршрутов разбить на две. В первой фиксируются данные маршрута целиком: номер, дата начала и конца, водитель и номер машины. Во второй таблице фиксируется порядок объезда клиентов по конкретным маршрутам. Очевидно, что в этом случае транзитивная зависимость исчезает.



НФБК для данной задачи будет выполняться автоматически, т.к. никаких детерминантов кроме первичных ключей нет.

Для проверки на четвертую нормальную форму особо следует проверить таблицу RouteItem, т.к. она является таблицей-прослойкой для организации

связи многие ко многим между таблицей маршрутов и клиентов. Пусть для проверки на многозначную зависимость в качестве таблицы A берется атрибут RouteNumber, а в качестве B и C атрибуты INN и ORDER_NUMBER соответственно. Очевидно, что ни о какой независимости этих атрибутов говорить не приходится, т.к. порядковый номер объезда клиента не может быть независимым от этого же клиента. Аналогичными рассуждениями убеждаемся, что если в качестве атрибутов A, B и C брать другие атрибуты, то многозначной зависимости все равно не возникнет.

Таким образом, итоговая структура базы данных соответствует определению четвертой нормальной формы.

3.6. Контрольные вопросы

1. Сколько всего существует нормальных форм?
2. В чем заключается сущность процесса нормализации?
3. В какой нормальной форме отсутствуют аномалии?
4. Почему на практике обычно ограничиваются приведением схемы базы данных только к третьей нормальной форме?

4. Лабораторная работа 4. Создание пустой базы данных

4.1. Цель работы

Изучить операции, проводимые с базами данных в целом. Получить навыки использования программ Power Architect и IB Expert" для создания, регистрации, подключения, извлечения метаданных СУБД Firebird. Изучить SQL-операторы для создания базы данных.

Время выполнения: 2 часа

4.2. Исходные данные

Исходными данными являются результаты предыдущей лабораторной работы.

4.3. Краткие теоретические сведения

Прямым проектированием называется процесс генерации физической схемы БД из логической модели.

Таблицы создаются командой CREATE TABLE, которая в основном определяет имя таблицы, в виде описания набора имен столбцов указан-

ных в определенном порядке. Она также определяет типы данных и размеры столбцов. Каждая таблица должна иметь по крайней мере один столбец.

Синтаксис команды CREATE TABLE:

CREATE TABLE <table-name >
(<column name > <data type>[(<size>)],
<column name > <data type> [(<size>)] ...);

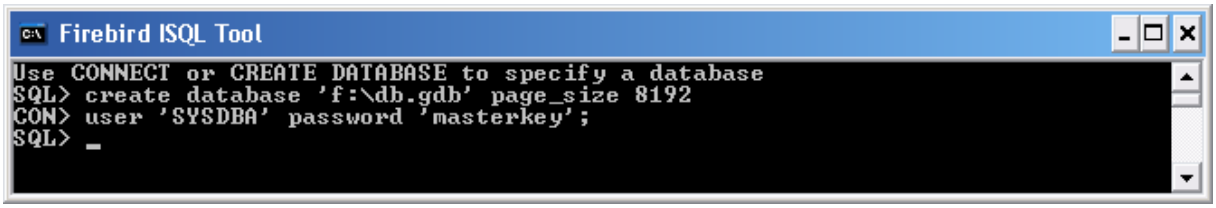
В различных СУБД присутствуют различные типы данных, поэтому детальный синтаксис команды не является универсальным.

4.4. Задание

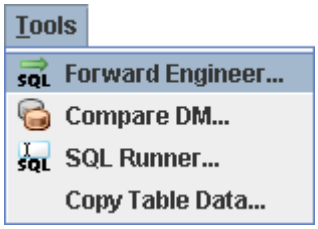
Средствами программы Power Architect создать SQL-скрипт для создания базы данных с пустыми таблицами, адаптировать его под СУБД Firebird (Power Architect не поддерживает данный вид СУБД). С помощью командной строки СУБД Firebird создать пустой файл базы данных. Далее с помощью программы IB Expert подключиться к этой базе данных и в ней выполнить адаптированный под Firebird SQL-скрипт.

4.5. Порядок выполнения работы

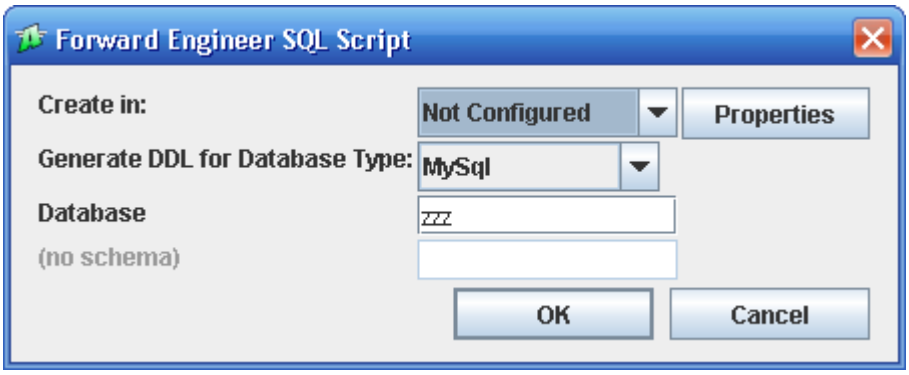
Для того, чтобы создать пустую базу данных сначала с помощью утилиты ISQL СУБД Firebird создадим файл базы данных.



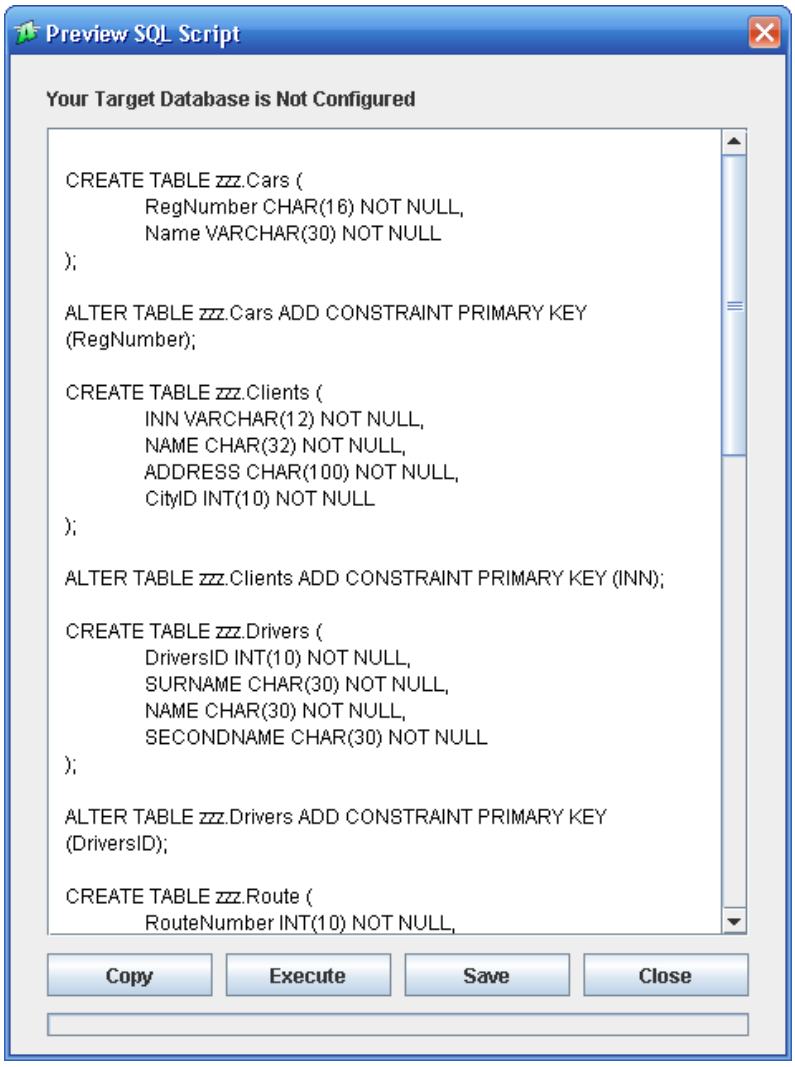
После выполнения этой операции в корне диска F должен появиться файл db.gdb. Далее с помощью опции Forward Engineer создадим SQL-скрипт для создания пустой базы данных.



Из-за того, что Power Architect не поддерживает создание СУБД Firebird, создадим скрипт для MySQL, а затем его отредактируем в соответствии с синтаксисом СУБД Firebird.



В качестве базы данных введем zzz. После нажатия на кнопку OK на экране появится скрипт для создания пустой базы данных в формате MySQL:

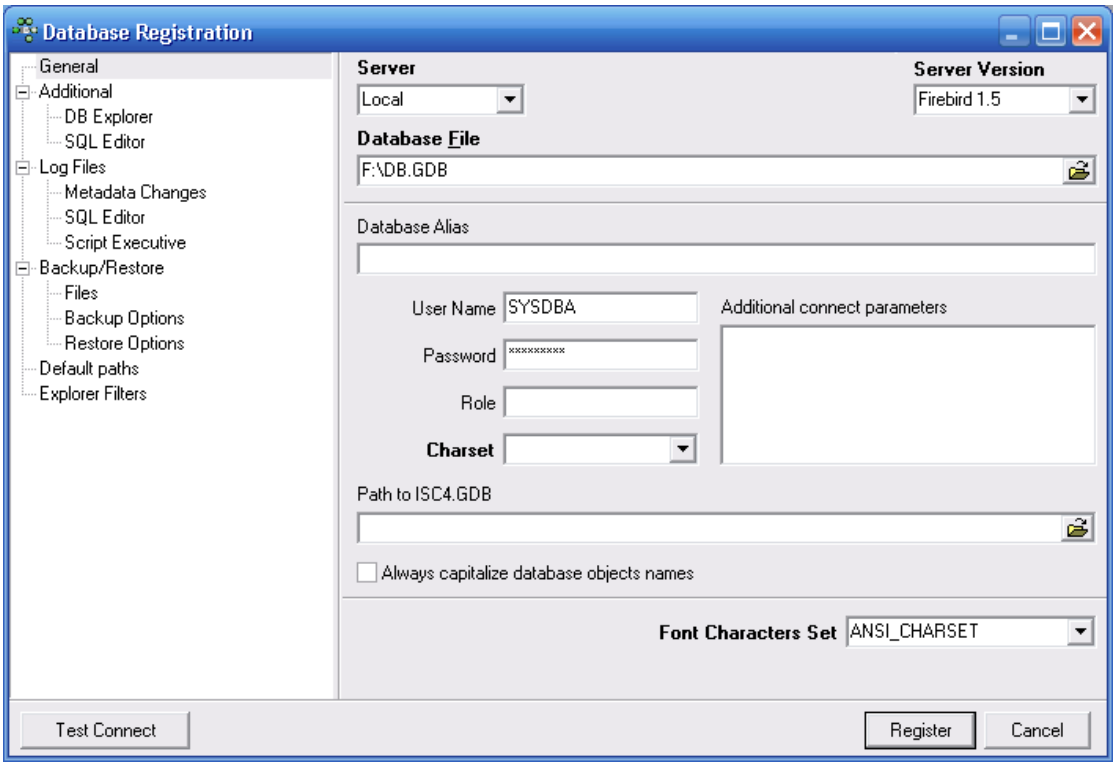


В случае, если при создании SQL-скрипта будут выдаваться ошибки, значит структура базы данных нуждается в доработке. Возможно какие-то таблицы содержат в своем названии запрещенные символы, какие-то связи между сущностями неправильно созданы и т.п.

Для того, чтобы адаптировать полученный скрипт под СУБД Firebird, из него необходимо убрать «zzz.», тип INT(10) заменить на INTEGER, а в конструкциях создания первичных ключей между ключевыми словами CONSTRAINT и PRIMARY KEY необходимо вставить уникальные имена, например следующим образом:

```
ALTER TABLE Cars ADD CONSTRAINT PK_CARS PRIMARY KEY
(RegNumber);
```

Далее откроем созданный файл базы данных в программе IB Expert.



В окне SQL Editor выполним отредактированный SQL-скрипт. Для того, чтобы исправить все возможные ошибки рекомендуется его выполнять пошагово, т.е. до ближайшей «;». Для выполнения команды необходимо нажать на кнопку  , а для того, чтобы сохранить изменения предназначена кнопка  .

4.6. Контрольные вопросы

1. Чем отличается прямое проектирование от обратного?
2. Опишите различные варианты создания пустой базы данных из логической модели.

5. *Лабораторная работа 5.* Заполнение данными

5.1. Цель работы

Изучить SQL-операторы по заполнению информации в базе данных
Время выполнения: 2 часа

5.2. Исходные данные

Исходными данными являются результаты предыдущей лабораторной работы.

5.3. Краткие теоретические сведения

Язык SQL является полнофункциональным языком манипулирования данными, который может использоваться не только для выборки из базы, но и для модификации ее содержимого. Операторы модификации информации в базе данных не столь сложны, как оператор SELECT. Для модификации содержимого базы данных в языке SQL предусмотрены три оператора:

- **INSERT** - предназначен для добавления данных в таблицу.
- **UPDATE** - предназначен для модификации уже помещенных в таблицу данных.
- **DELETE** - позволяет удалять из таблицы строки данных.

Добавление новых данных в таблицу (оператор INSERT)

Существует две формы оператора INSERT. Первая предназначена для вставки единственной строки в указанную таблицу. Эта форма оператора имеет следующий формат:

```
INSERT INTO table name [(column_list)]
VALUES (data_value_list)
```

Здесь параметр column_list представляет собой список, состоящий из имен одного или более столбцов, разделенных запятыми. Данный параметр является необязательным. Если он опущен, то предполагается использование списка из имен всех столбцов таблицы. Параметр data_value_list (список значений данных) должен соответствовать параметру colum_list следующим образом:

- количество элементов в обоих списках должно быть одинаковым;
- должно существовать прямое соответствие между позицией одного и того же элемента в обоих списках,
- типы данных элементов списка data_value_list должны быть совместимы с типом данных соответствующих столбцов таблицы.

Вторая форма оператора INSERT позволяет скопировать множество строк одной таблицы в другую таблицу. Этот оператор имеет следующий формат:

```
INSERT INTO table_name [(column_list)]
SELECT ...
```

Здесь параметры table_name и column_list имеют тот же формат и смысл, что и при вставке в таблицу одной строки. Предложение SELECT может представлять собой любой допустимый запрос. Строки, вставляемые в указанную таблицу, в точности соответствуют строкам результирующей таблицы, созданной при выполнении вложенного запроса. Все ограничения, указанные выше для первой формы оператора INSERT, применимы и в этом случае.

Модификация данных в базе (оператор UPDATE)

Оператор UPDATE позволяет изменять содержимое уже существующих строк указанной таблицы. Этот оператор имеет следующий формат:

```
UPDATE table_name
SET column_name1 = data_value1
[, column_name2=data_value2 ...]
[WHERE search_condition]
```

В предложении SET необходимо указывать имена одного или более столбцов, данные в которых необходимо изменить. Предложение WHERE является необязательным. Если оно опущено, значения указанных столбцов будут изменены во всех строках таблицы. Если предложение WHERE присутствует, обновлены будут только те строки, которые удовлетворяют условию поиска, заданному в параметре search_condition. Параметры data_value представляют новые значения соответствующих столбцов и должны быть совместимы с ними по типу данных.

Удаление данных из базы (оператор DELETE)

Оператор DELETE позволяет удалять строки данных из указанной таблицы, данный оператор имеет следующий формат:

```
DELETE FROM table_name [WHERE search_condition]
```

Параметр search_condition является необязательным - если он опущен, из таблицы будут удалены все существующие в ней строки. Однако сама по себе таблица удалена не будет. Если предложение WHERE присутствует, то из таблицы будут удалены только те строки, которые удовлетворяют условию отбора, заданному параметром search_condition.

5.4. Задание

Написать SQL-скрипт, который заполняет данными все таблицы схемы. Данный скрипт должен выполняться многократно, т.е. он должен предварительно очищать все таблицы, а затем создавать в них новые записи.

5.5. Порядок выполнения работы

Для тестирования скрипта заполнения будем использовать окно Script Executive. Добавление данных в таблицы необходимо производить с тех, которые не ссылаются на другие таблицы, затем тех, которые ссылаются на уже заполненные и т.д. Удаление данных необходимо производить в обратном порядке.

Создадим скрипт, который удаляет все данные и добавляет в таблицы двух водителей, двух автомобилей, четырех клиентов из Москвы и Самыра и два маршрута их объезда.

```
DELETE FROM RouteItem;
DELETE FROM Route;
DELETE FROM Clients;
DELETE FROM City;
DELETE FROM Cars;
DELETE FROM Drivers;

INSERT INTO Drivers (DRIVERSID, SURNAME, NAME, SECONDNAME)
VALUES (1, 'Иванов', 'Иван', 'Иванович');
INSERT INTO Drivers (DRIVERSID, SURNAME, NAME, SECONDNAME)
VALUES (2, 'Петров', 'Петр', 'Петрович');

INSERT INTO Cars (REGNUMBER, NAME) VALUES ('a762вв163', 'Камаз');
INSERT INTO Cars (REGNUMBER, NAME) VALUES ('c762вн163', 'Мерседес');

INSERT INTO City (CITYID, NAME) VALUES (1, 'Самара');
INSERT INTO City (CITYID, NAME) VALUES (2, 'Москва');
```

```
INSERT INTO Clients (INN, NAME, ADDRESS, CITYID)
VALUES ('632000000001', 'ООО "Тонус"', 'Ленинградская 1', 1);
INSERT INTO Clients (INN, NAME, ADDRESS, CITYID)
VALUES ('632000000002', 'ООО "СТЕР"', 'Потапова 2', 1);
INSERT INTO Clients (INN, NAME, ADDRESS, CITYID)
VALUES ('632000000003', 'ООО "Карго"', 'Урицкого 3-1', 2);
INSERT INTO Clients (INN, NAME, ADDRESS, CITYID)
VALUES ('632000000004', 'ООО "Пронто"', 'Урицкого 5', 2);

INSERT INTO ROUTE (RouteNumber, DATE_FROM, DATE_TO, NAME,
DriversID, RegNumber)
VALUES (1, '01.02.2009', '01.02.2009', 'Самарский', 1, 'а762вв163');
INSERT INTO ROUTE (RouteNumber, DATE_FROM, DATE_TO, NAME,
DriversID, RegNumber)
VALUES (2, '01.02.2009', '05.02.2009', 'Московский', 2, 'с762вн163');

INSERT INTO ROUTEITEM (RouteNumber, INN, ORDER_NUMBER)
VALUES (1, '632000000001', 1);
INSERT INTO ROUTEITEM (RouteNumber, INN, ORDER_NUMBER)
VALUES (1, '632000000002', 2);
INSERT INTO ROUTEITEM (RouteNumber, INN, ORDER_NUMBER)
VALUES (2, '632000000003', 1);
INSERT INTO ROUTEITEM (RouteNumber, INN, ORDER_NUMBER)
VALUES (2, '632000000004', 2);

COMMIT;
```

5.6. Контрольные вопросы

1. Какие команды в языке SQL существуют для заполнения информацией базу данных?
2. Приведите пример любой команды по заполнению информацией базы данных, которая при внешнем правильном синтаксисе при выполнении выдаст ошибку.

6. Лабораторная работа 6. Знакомство с языком SQL

6.1. Цель работы

Изучить основные возможности языка SQL для извлечения информации из базы данных

Время выполнения: 4 часа

6.2. Исходные данные

Исходными данными являются индивидуальное задание и результаты предыдущей лабораторной работы.

6.3. Краткие теоретические сведения

Все запросы в SQL состоят из одиночной команды. Структура этой команды обманчиво проста, потому что вы должны расширять ее так, чтобы выполнить высоко сложные оценки и обработки данных. Эта команда называется - SELECT (ВЫБОР). Синтаксис данной команды имеет вид:

```
SELECT [distinct|All]{*|[column_i [as name_i]][,...]}
FROM table_name [alias] [, ...]
[WHERE condition]
[GROUP BY group_column_list] [HAVING condition]
[ORDER BY order_column_list]
```

Ниже представлено краткое описание основных секций команды SQL (см. таблицу 1).

Таблица 1

Краткое описание основных секций команды SQL

Секция	Описание
FROM	Определяются имена используемой таблицы или нескольких таблиц
WHERE	Выполняется фильтрация строк объекта в соответствии с заданными условиями
GROUP BY	Образуются группы строк, имеющих одно и то же значение в указанном столбце
HAVING	Фильтруются группы строк объекта в соответствии с указанным условием
SELECT	Устанавливается, какие столбцы должны присутствовать в выходных данных
ORDER BY	Определяется упорядоченность результатов выполнения оператора

Выбор строк (предложение WHERE)

В приведенных выше примерах в результате выполнения операторов SELECT выбирались все строки указанной таблицы. Однако в большинстве случаев требуется тем или иным образом ограничить набор строк, помещаемых в результирующую таблицу запроса. Это достигается с помощью указания в запросе предложения WHERE. Оно состоит из ключевого слова, за которым следует перечень условий поиска, определяющих те строки, которые должны быть выбраны при выполнении запроса. Существует пять основных типов условий поиска:

- ***Сравнение***. Сравниваются результаты вычисления одного выражения с результатами вычисления другого выражения.
- ***Диапазон***. Проверяется, попадает ли результат вычисления выражения в заданный диапазон значений.
- ***Принадлежность к множеству***. Проверяется, принадлежит ли результат вычисления выражения заданному множеству значений.
- ***Соответствие шаблону***. Проверяется, отвечает ли некоторое строковое значение заданному шаблону.
- ***Значение NULL***. Проверяется, содержит ли данный столбец определитель NULL (пустое значение).

Различные условия на выборку данных могут объединяться друг с другом с помощью логических операторов NOT, AND и OR[1]. С помощью скобок можно изменить порядок выполнения условий в запросе. Вычисление выражений в условиях выполняется по следующим правилам:

- Выражение вычисляется слева направо.
- Первыми вычисляются подвыражения в скобках.
- Операторы NOT выполняются до выполнения операторов AND и OR.
- Операторы AND выполняются до выполнения операторов OR.

Для устранения любой возможной неоднозначности рекомендуется использовать скобки. Рассмотрим примеры использования всех указанных типов условий поиска.

Условие типа «Сравнение»

В языке SQL используются следующие операторы сравнения:

=	равенство
<	меньше
>	больше
<=	меньше или равно

[1] Логические операторы приведены в порядке убывания их приоритета. Кроме этого приоритет самих логических операторов ниже, чем приоритет какого-либо условия поиска, поэтому нет необходимости различные логические условия заключать в скобки.

>= больше или равно
<> не равно (стандарт ISO)
!= не равно (используется в некоторых диалектах)

Например, если необходимо вывести все товары из прайс-листа с закупочной ценой меньшей 10000 единиц, которой имеется на складе в количестве, большем 100 штук, то потребуется выполнить следующий запрос:

```
SELECT * FROM Прайс-лист
WHERE Закупочная_цена<10000 AND Количество>100
```

Условие типа «Диапазон»

Данный тип условия реализуется с помощью условий (BETWEEN/NOT BETWEEN). Например, если мы хотим вывести из прайс-листа все товары с кредитной ценой от 10000 до 20000, то можно воспользоваться следующим запросом:

```
SELECT * FROM Прайс-лист
WHERE Цена_в_кредит BETWEEN 10000 AND 20000
```

Наличие ключевого слова BETWEEN и соответствующей проверки лишь незначительно повышает выразительную мощность языка SQL, поскольку те же самые результаты могут быть достигнуты с помощью выполнения двух обычных проверок. Так, приведенный выше запрос можно переписать следующим образом:

```
SELECT * FROM Прайс-лист
WHERE Цена_в_кредит>=10000 AND
      Цена_в_кредит<=20000
```

Однако проверка вхождения в диапазон с помощью ключевого слова BETWEEN является более простым способом записи условий выборки, чем обычные проверки.

Условие типа «Принадлежность к множеству»

Данный тип условия реализуется с помощью условий (IN/NOT IN). Например, если мы хотим вывести всех студентов с именами Иван и Петр, то можно воспользоваться следующим запросом:

```
SELECT * FROM Студенты
WHERE Имя IN ('Иван', 'Петр')
```

Как и в случае ключевого слова BETWEEN, условие IN незначительно повышает выразительную мощность языка SQL, так как тот же самый запрос может быть переписан следующим образом:

```
SELECT * FROM Студенты
WHERE Имя='Иван' OR Имя='Петр'
```

Однако использование ключевого слова IN представляет собой более эффективный способ записи условий поиска, особенно если набор допустимых значений достаточно велик.

Условие типа «Соответствие шаблону»

Данный тип условия реализуется с помощью условий (LIKE/NOT LIKE). Он используется в случае, если невозможно точно указать значение поиска полностью, но можно задать какие-либо его части. В языке SQL существует два специальных символа шаблона, используемых при проверке значений:

%	Символ процента представляет собой произвольную последовательность из нуля и более символов
_	Символ подчеркивания представляет любой одиночный символ

Так, например, если мы хотим вывести всех студентов, фамилия которых начинается на ‘Ива’, вторая буква имени – буква ‘А’, а отчество оканчивается на ‘ИЧ’, то можно воспользоваться следующим запросом:

```
SELECT * FROM Студенты
WHERE Фамилия  LIKE 'Ива%' AND
      Имя      LIKE '_А%'  AND
      Отчество LIKE '%ИЧ'
```

Условие типа «Значение NULL»

С помощью условия данного типа реализуется проверка, содержит ли какой-либо атрибут пустые значения. Например, если мы хотим вывести всех студентов с незаполненным именем, необходим следующий запрос:

SELECT * FROM Студенты **WHERE** Имя **IS NULL**

Сортировка результатов (предложение ORDER BY)

Для сортировки результатов выборки в оператор SELECT помещается фраза ORDER BY. Она включает список идентификаторов столбцов, разделенных запятыми, по которым требуется упорядочить результирующую

таблицу запроса. Идентификатор столбца может представлять собой либо его имя, либо номер, который идентифицирует элемент списка SELECT его позицией в этом списке. Самый левый элемент списка имеет номер 1, следующий – номер 2 и т.д.[2]

Так, если требуется упорядочить список студентов по фамилии, то можно использовать следующий SQL:

```
SELECT Фамилия,Имя,Отчество FROM Студенты
ORDER BY Фамилия
```

или

```
SELECT Фамилия,Имя,Отчество FROM Студенты
ORDER BY 1
```

Фраза ORDER BY позволяет упорядочить выбранные записи в порядке возрастания (ASC) или убывания (DESC) значений любого столбца или комбинации столбцов, независимо от того, присутствуют эти столбцы в таблице результатов или нет. Фраза ORDER BY всегда должна быть последним элементом в операторе SELECT.

Обобщающие функции

В стандарте языка SQL имеется пять обобщающих функций:

COUNT	Возвращает количество значений в указанном столбце
SUM	Возвращает сумму значений в указанном столбце
AVG	Возвращает усредненное значение в указанном столбце
MIN	Возвращает минимальное значение в указанном столбце
MAX	Возвращает максимальное значение в указанном столбце

Все эти функции оперируют со значениями в единственном столбце таблицы и возвращают единственное значение. Функции COUNT, MIN и MAX применимы как к числовым, так и к нечисловым полям, тогда как функции SUM и AVG могут использоваться только в случае числовых полей. За исключением COUNT(*), при вычислении результатов любых функции сначала

[2] Номера столбцов являются функцией, отвергнутой стандартом ISO, кроме этого жесткая привязка к номеру столбца является источником потенциальной ошибки при частой правке SQL-запросов. По этим причинам использование номеров столбцов при сортировке не рекомендуется.

исключаются все пустые значения, после чего требуемая операция применяется только к оставшимся конкретным значениям столбца.

Обобщающие функции могут использоваться только списке предложения SELECT и в составе предложения HAVING. Во всех других случаях использование этих функций недопустимо.

Например, если необходимо определить количество студентов с именем Иван, то можно воспользоваться следующим запросом:

```
SELECT count(*) FROM Студенты where имя='Иван'
```

Группировка результатов (предложение GROUP BY)

Приведенный выше пример сводных данных по студентам подобен итоговым строкам, обычно размещаемым в конце отчетов. В них все детальные данные отчета сжимаются в одну обобщающую строку. Однако очень часто в отчетах требуется формировать и промежуточные итоги. Для этой цели в операторе SELECT может указываться фраза GROUP BY. Запрос, в котором присутствует фраза GROUP BY, называется **группирующим запросом**, поскольку в нем группируются данные, полученные в результате выполнения операции SELECT, после чего для каждой отдельной группы создается единственная суммарная строка. Столбцы, перечисленные во фразе GROUP BY, называются **группируемыми столбцами**. Стандарт ISO требует, чтобы предложение SELECT и фраза GROUP BY были тесно связаны между собой. При использовании в операторе SELECT фразы GROUP BY каждый элемент списка в предложении SELECT должен иметь **единственное значение для всей группы**. Более того, предложение SELECT может включать только следующие типы элементов:

- имена столбцов;
- обобщающие функции;
- константы;
- выражения из комбинации перечисленных выше элементов.

Все имена столбцов, приведенные в списке предложения SELECT, должны присутствовать и во фразе GROUP BY - за исключением случаев, когда имя столбца используется в обобщающей функции. Обратное правило не является справедливым – во фразе GROUP BY могут присутствовать имена столбцов, отсутствующие в списке предложения SELECT. Если совместно с фразой GROUP BY используется предложение WHERE, то оно обрабатывается первым, а группировке подвергаются только те строки, которые удовлетворяют условию поиска.

Стандартом ISO определено, что при проведении группировки все отсутствующие значения рассматриваются как равные. Если две строки таблицы в одном и том же группируемом столбце содержат значения NULL, а

значения во всех остальных непустых группируемых столбцах идентичны, то они помещаются в одну и ту же группу.

Если необходимо вывести список всех номеров накладных с указанием суммы заказа из таблицы «Заказы_по_накладным», то для этого можно использовать следующий запрос:

```
SELECT Код_накладной, sum(Цена*Количество)
FROM Заказы_по_накладным
GROUP BY Код_накладной
ORDER BY Код_накладной
```

Ограничения на выполнение группирования (предложение HAVING)

Если в предложении GROUP BY необходимо задать какие-либо дополнительные ограничения на отбор тех групп, которые будут помещены в результирующую таблицу запроса, то для этого необходимо использовать внутри ORDER BY предложение HAVING. Хотя данная фраза и предложение WHERE имеют сходный синтаксис, их назначение различно. Предложение WHERE предназначено для фильтрации отдельных строк, тогда как фраза HAVING используется для фильтрации групп, помещаемых в результирующую таблицу запроса. Стандарт ISO требует, чтобы имена столбцов, используемые во фразе HAVING, обязательно присутствовали в списке фразы GROUP BY или применялись в обобщающих функциях. На практике условия поиска во фразе HAVING всегда включают, по меньшей мере, одну обобщающую функцию, в противном случае эти условия поиска должны быть помещены в предложение WHERE и применяться для отбора отдельных строк[3].

Фраза HAVING не является необходимой частью языка SQL - любой запрос, написанный с использованием фразы HAVING, может быть представлен в ином виде, без ее применения.

Если в предыдущем случае необходимо вывести список только тех накладных, в заказ которых входило не менее 10 наименований продукции, необходимо предыдущий запрос немного изменить:

```
SELECT Код_накладной, sum(Цена*Количество)
FROM Заказы_по_накладным
GROUP BY Код_накладной
```

[3] Не забывайте, что обобщающие функции не могут использоваться в предложении WHERE.

```
HAVING count(*)>10
ORDER BY Код_накладной
```

Подзапросы

Подзапросами называются законченные операторы SELECT, внедренные в тело другого оператора SELECT. Внешний оператор SELECT использует результат выполнения внутреннего оператора для определения содержания окончательного результата всей операции. Внутренние запросы могут быть помещены в предложения WHERE и HAVING внешнего оператора SELECT – в этом они получают название подзапросов, или вложенных запросов. Кроме того, внутренние операторы SELECT могут использоваться в операторах INSERT, UPDATE и DELETE[4]

. Существует три типа подзапросов.

- ***Скалярный подзапрос*** возвращает единственное значение, выбираемое из пересечения одного столбца с одной строкой.
- ***Строковый подзапрос*** возвращает значения нескольких столбцов таблицы, но в виде единственной строки. Строковый подзапрос может использоваться везде, где применяется конструктор строковых значений – обычно это предикаты.
- ***Табличный подзапрос*** возвращает значения одного или больше столбцов таблицы, размещенные в более чем одной строке. Табличный подзапрос может использоваться везде, где допускается указывать таблицу – например, как операнд предиката IN.

6.4. Задание

Написать четыре SQL-запроса к разработанной базе данных, предложенные преподавателем

6.5. Порядок выполнения работы

Запустив IB Expert, необходимо соединиться с созданной ранее базой данных, открыть окно SQL Editor и отладить предложенные запросы.

1. Пусть требуется вывести всех водителей, фамилия которых оканчивается на «ов», в имя которых «Иван», а отчество заполнено.

[4] Данные SQL-операторы предназначены для модификации информации в базе данных.

```
SELECT * FROM Drivers
WHERE SURNAME LIKE “%ОВ” AND NAME=”ИВАН” AND
        SECONDNAME IS NOT NULL
```

2. Вывести водителей, которые ездят по маршрутам в Самаре

```
SELECT DISTINCT D.* FROM Drivers D
INNER JOIN Route R ON R.DriversID=D.DriversID
INNER JOIN RouteItem RI ON  RI.RouteNumber=R.RouteNumber
INNER JOIN Clients C ON C.INN = RI.INN
INNER JOIN City Ci ON Ci.CityID = C.CityID
WHERE Ci.NAME='Самара'
```

3. Вывести все маршруты, которые идут только через двух клиентов

```
SELECT R.RouteNumber, COUNT(*)
FROM Route R
INNER JOIN RouteItem RI ON  RI.RouteNumber=R.RouteNumber
INNER JOIN Clients C ON C.INN = RI.INN
GROUP BY R.RouteNumber
HAVING COUNT(*)=2
```

4. Вывести сводную информацию: ФИО водителя, на скольких автомобилях ездил и сколько раз выезжал на маршрут

```
SELECT D.SURNAME, D.NAME, D.SECONDNAME,
    (
        SELECT COUNT(*)
        FROM Cars C
        WHERE EXISTS(
            SELECT 1 FROM Route R
            WHERE R.RegNumber=C.RegNumber AND
                R.driversid=D.driversid
        )
    ) Cars_Count,
    (
        SELECT COUNT(*)
        FROM Route R
        WHERE R.driversid=D.driversid
    ) Route_Count
FROM Drivers D
```

5. Написать два варианта запроса, выводящего все автомашины, на которых ездил водитель Петров.

Первый вариант:

```
SELECT DISTINCT C.*
FROM Cars C
INNER JOIN Route R ON R.RegNumber=C.RegNumber
INNER JOIN Drivers D ON D.DriversID=R.DriversID
WHERE D.SURNAME='Петров'
```

Второй вариант:

```
SELECT C.*
FROM Cars C
WHERE EXISTS (
        SELECT 1
        FROM Route R
        INNER JOIN Drivers D ON D.DriversID=R.DriversID
        WHERE R.RegNumber=C.RegNumber AND
                D.SURNAME='Петров'
)
```

6.6. Контрольные вопросы

1. Приведите пример использования команды SELECT для поиска информации в нескольких таблицах
2. Для чего нужна команда LIKE
3. Приведите пример запросов, которые извлекают одну и ту же информацию различными способами.

7. Лабораторная работа 7. Оптимизация запросов

7.1. Цель работы

Научиться читать планы исполнения запросов и получить представление о способах повышения скорости выполнения запросов

Время выполнения: 2 часа

7.2. Исходные данные

Исходными данными являются результаты предыдущей лабораторной работы.

7.3. Краткие теоретические сведения

Для оптимизации запросов большую помощь может оказать анализ плана его исполнения. Все производители серверов баз данных предлагают специальные инструменты создания планов исполнения запросов, зачастую обладающие удобным графическим интерфейсом.

К общим рекомендациям по оптимизации SQL-запросов можно отнести следующие:

- использовать правильный индекс;
- запретить использование неподходящих индексов;
- использовать желаемый порядок соединения;
- запретить соединения в неправильном порядке;
- выбрать порядок выполнения внешних запросов и подзапросов;
- предоставить оптимизатору верные данные для анализа.

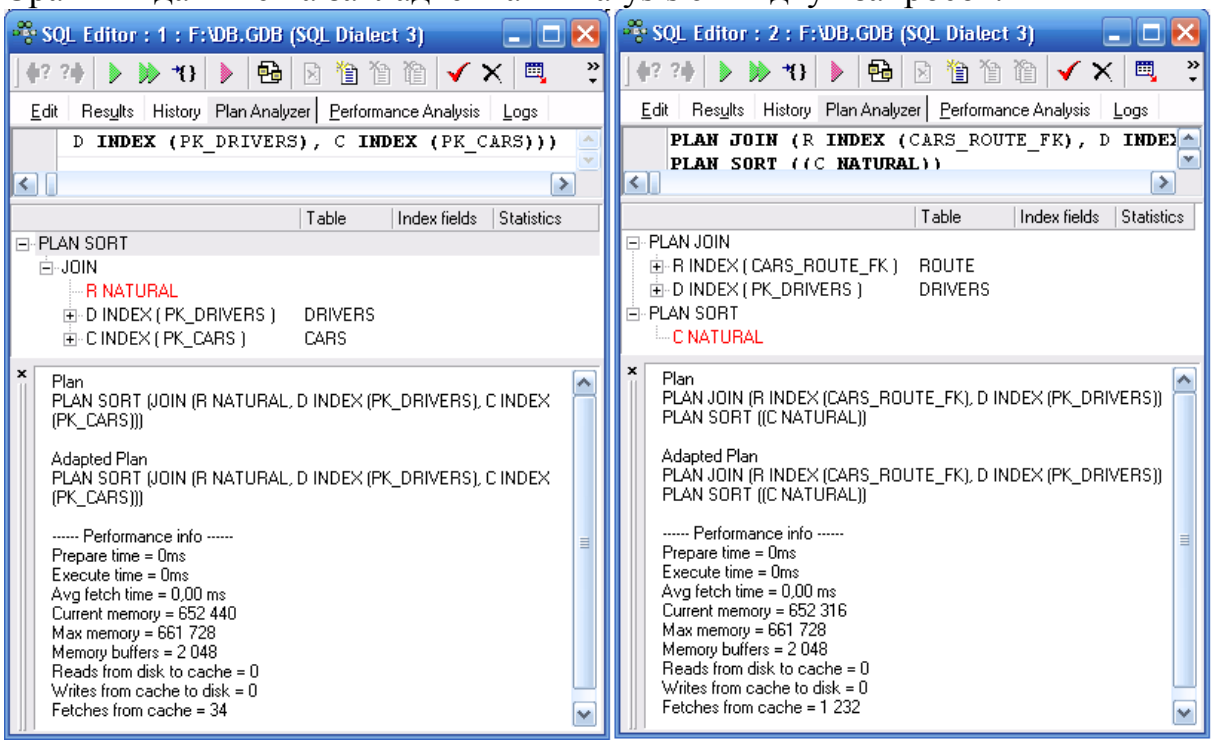
7.4. Задание

Провести анализ SQL-запросов, разработанных в предыдущей работе, определить способы повышения их быстродействия и реализовать соответствующих комплекс мер.

7.5. Порядок выполнения работы

Разработаем план оптимизации на примере двух вариантов пятого запроса предыдущей лабораторной работы.

Сравним данные на закладке Plan Analysis этих двух запросов.



Внешне запросы выполняются примерно одинаковым способом. Однако, переключившись на закладку Performance Analysis можно видеть, что параметр NIR (неиндексированное чтение) второго запроса просто огромен (302 против 0 у первого запроса).

Enhanced Info

Table name	IR	NIR	UPD	DEL	INS
CARS	2	0	0	0	0
DRIVERS	3	0	0	0	0
ROUTE	0	3	0	0	0

Enhanced Info

Table name	IR	NIR	UPD	DEL	INS
CARS	0	302	0	0	0
DRIVERS	2	0	0	0	0
ROUTE	2	0	0	0	0

Попробуем повысить производительность за счет создания индексов. В СУБД Firebird автоматически создаются индексы по ссылочным полям. Создадим индекс по полю фамилия в таблице водителей, т.к. в запросе это поле используется. Для этого выполним команду:

```
CREATE INDEX IDX_DRIVERS_SURNAME ON DRIVERS
(SURNAME)
```

После этого данные закладки Performance Analysis принимают следующий вид:

Enhanced Info

Table name	IR	NIR	UPD	DEL	INS
CARS	2	0	0	0	0
DRIVERS	2	0	0	0	0
ROUTE	2	0	0	0	0

Enhanced Info

Table name	IR	NIR	UPD	DEL	INS
CARS	0	302	0	0	0
DRIVERS	2	0	0	0	0
ROUTE	2	0	0	0	0

Таким образом установлено, что в результате произведенных мероприятий производительность первого запроса повысилась, а второго не изменилась.

7.6. Контрольные вопросы

1. Что такое план исполнения запросов?
2. По каким признакам можно отличить «хороший» план исполнения от «плохого»?

8. Лабораторная работа 8. Защита баз данных

8.1. Цель работы

Изучить используемые в Firebird способы обеспечения безопасности данных. Получить навыки создания и удаления пользователей и ролей, а также использования команд выдачи и отмены прав. Изучение возможности построения системы защиты от НСД с помощью представлений.

Время выполнения: 2 часа

8.2. Исходные данные

Исходными данными являются результаты предыдущей лабораторной работы.

8.3. Краткие теоретические сведения

Обеспечение безопасности хранимых данных является неотъемлемой частью любой современной СУБД. В большинстве СУБД защита данных основана на концепции пользователей, которые получают определенные права для работы с объектами базы данных. Под пользователем понимается регистрационная запись, состоящая из имени пользователя и его пароля.

Администратор СУБД Firebird (пользователь SYSDBA) заводит необходимое число пользователей и назначает им необходимые для выполнения их должностных обязанностей права.

В СУБД Firebird данные о пользователях всех баз данных хранятся не в этих базах данных, а в особой базе данных пользователей, которая располагается в файле security.fdb.

Информация в самих базах данных никак не шифруется и не защищается. Решение о разрешении доступа пользователю к определенному объекту базы данных принимает сервер СУБД путем сравнения прав, выданных на этот объект, с правами, которые имеет данный пользователь.

Следствием этого является то, что, физически скопировав базу данных на компьютер с другим сервером Firebird, можно воспользоваться паролем администратора этого сервера и получить полный доступ к информации в базе данных. Для обеспечения безопасности информации, хранящейся в базе данных Firebird, следует осуществить защиту файла базы данных на уровне ОС, запретив сетевой доступ к файлам базы данных и установив соответствующие права доступа на папки и файлы, а также ограничить доступ посторонних лиц к компьютеру-серверу.

Каждый пользователь в SQL-базе данных имеет набор привилегий. Это то, что пользователю разрешается делать. Эти привилегии могут изменяться со временем – новые добавляться, старые удаляться. Некоторые из

этих привилегий определены в ANSI SQL, но имеются и дополнительные привилегии, которые являются также необходимыми.

SQL привилегии, определенные в ANSI – это привилегии объекта. Это означает, что пользователь имеет привилегию, чтобы выполнить данную команду только на определенном объекте в базе данных. Привилегии объекта связаны одновременно и с пользователями, и с таблицами. То есть, привилегия дается определенному пользователю в указанной таблице, или базовой таблице или представлении. Пользователь, создавший таблицу, является владельцем этой таблицы. Это означает, что пользователь имеет все привилегии в этой таблице и может передавать привилегии другим пользователям в этой таблице.

Привилегии, которые можно назначить пользователю:

SELECT	Возможность выполнять запросы в таблице
INSERT	Возможность вставлять записи в таблицу
UPDATE	Возможность изменять записи в таблице
DELETE	Возможность удалять записи в таблице
REFERENCES	Возможность создавать внешний ключ на таблицу[5]

Для назначения пользователям привилегий, в языке SQL предназначена команда **GRANT**. Например, если необходимо разрешить пользователю Иванову обращаться к таблице заказов, то потребуется следующий оператор:

```
GRANT SELECT ON Заказы TO Иванов;
```

Когда SQL получает команду GRANT, он проверяет привилегии пользователя, подавшего эту команду, чтобы определить, допустима ли эта команда. Сам Иванов, естественно, самостоятельно не может выдать эту команду. Для того, чтобы разрешить Иванову также редактировать записи в таблице заказов, потребуется выполнить следующий SQL-оператор:

GRANT UPDATE ON Заказы **TO** Иванов;

[5] Кроме этого существует ряд привилегий, которые не являются стандартными, например, право создавать индекс, изменять структуру таблицы и т.п. Для того чтобы узнать весь набор привилегий конкретной СУБД необходимо тщательно изучить специфическую техническую литературу, желательно непосредственно от самого разработчика СУБД.

Однако, если необходимо ограничить Иванова в изменении только количества заказанного товара и наименования товара, потребуется небольшая корректировка оператора:

GRANT UPDATE (Кол-во, Наим_товара) **ON** Заказы **TO** Иванов;

Иногда, создателю таблицы хочется, чтобы другие пользователи могли получить привилегии в его таблице. Обычно это делается в системах, где один или более разработчиков создают несколько базовых таблиц, а затем передают ответственность за них тем, кто будет фактически с ними работать. SQL позволяет делать это с помощью предложения WITH GRANT OPTION. Если необходимо дать Иванову право предоставлять привилегию SELECT в таблице заказов другим пользователям, необходимо выполнить оператор:

```
GRANT SELECT ON Заказы TO Иванов WITH GRANT OPTION;
```

Для отмены привилегий предназначена команда **REVOKE**. Она похожа по синтаксису на команду GRANT, но имеет обратный смысл. Чтобы удалить привилегию INSERT для Иванова в таблице Заказов, можно ввести

```
REVOKE INSERT ON Заказы FROM Иванов;
```

Кроме этого, для ограничения доступа удобно использовать представления.

Представление – тип таблицы, чье содержимое выбирается из других таблиц с помощью выполнения SQL-запроса.

Данные представлений можно использовать в запросах абсолютно также как и данные таблиц, для пользователя не будет никаких внешних различий. Представления подобны окнам, которые из всей информации, находящейся в базе данных выводят только то, что какой-либо пользователь имеет желание или полномочие видеть. Таким образом, с помощью представлений можно построить хотя и простую, но при определенных условиях, достаточно мощную защиту от несанкционированного доступа к базе данных.

8.4. Задание

Создать три роли. Первая должна иметь полный доступ ко всем объектам. Второй должен быть запрещен доступ на просмотр/редактирование

(по выбору) к какому-либо объекту базы данных. Третья роль должна быть ограничена к некоторому подмножеству записей какой-либо таблицы. Создать трех пользователей, которые обладают вышеперечисленными ролями соответственно.

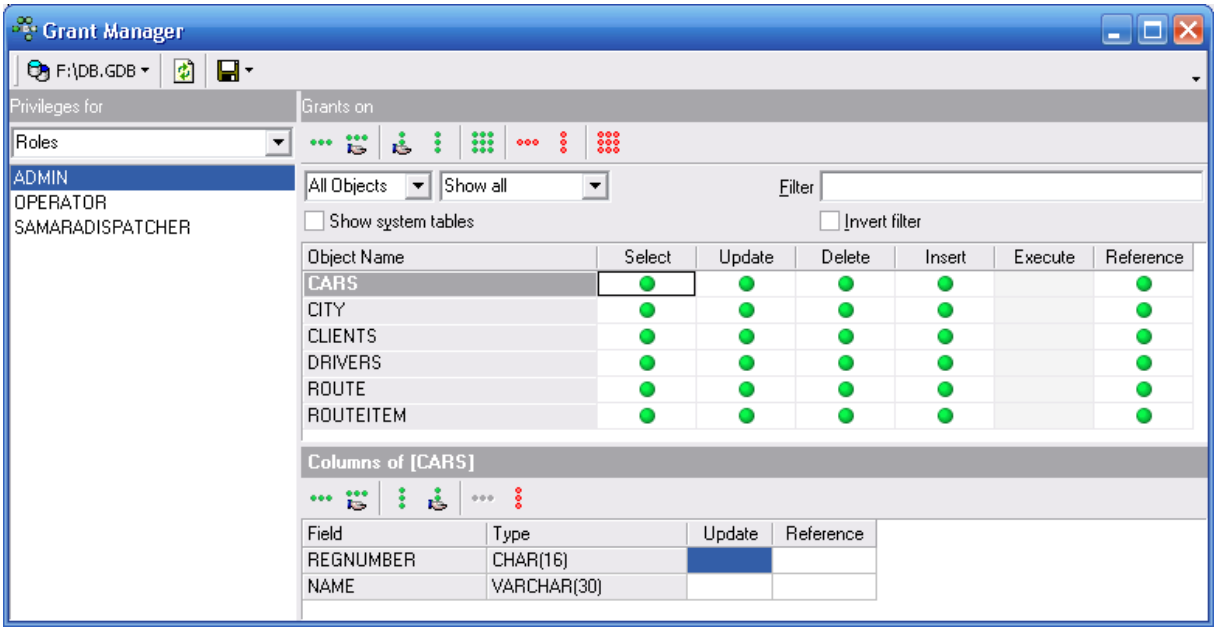
8.5. Порядок выполнения работы

Для создания роли в диалоговом режиме программы «IB Expert» необходимо выполнить следующие действия:

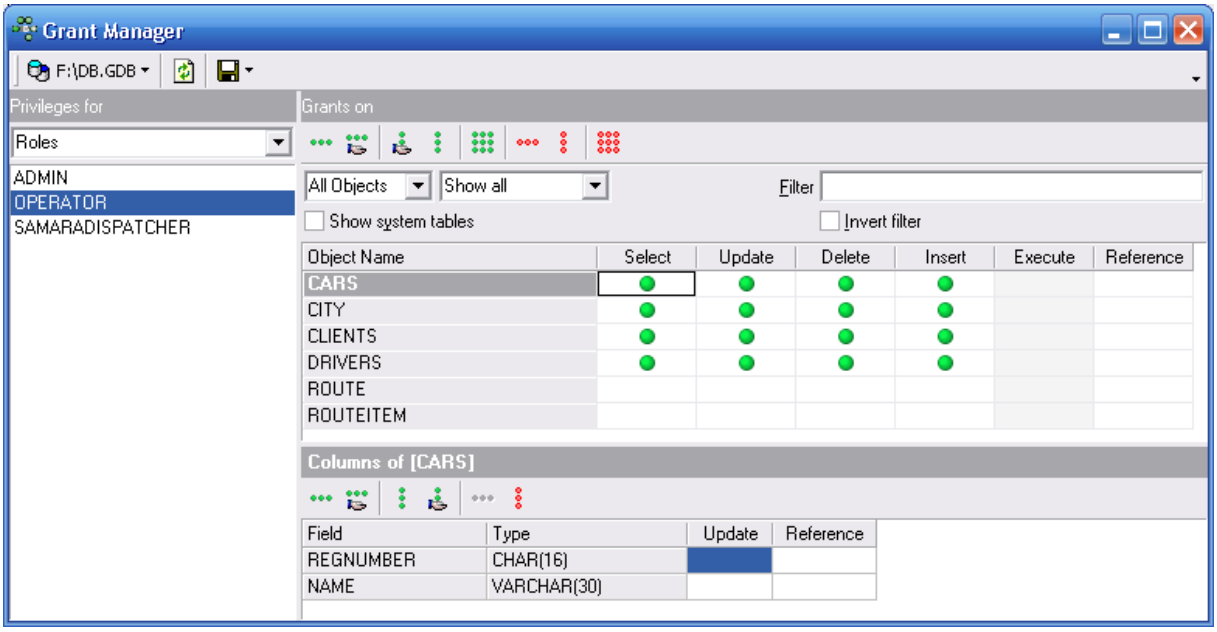
1. Подключиться к базе данных.
2. Выполнить команду "Database->New Role".
3. Ввести имя роли и нажать кнопку [OK].

Создадим вышеописанным способом три роли: Admin, Operator и SamaraDispatcher. Первая роль будет обладать полномочиями доступа на просмотр/вставку/изменение/удаление из любой таблицы. Вторая роль получает доступ на просмотр/вставку/изменение/удаление данных в таблицы клиентов, водителей, городов и автомобилей. Третья роль пусть обладает полномочиями просмотра/вставки в таблицы маршрутов (Route) и порядка объезда клиентов (RouteItem). К остальным таблицам третья роль получает доступ только на чтение, однако в справочнике клиентов она видит только клиентов, работающих в Самаре и соответственно, только этих клиентов может добавить в маршрут объезда.

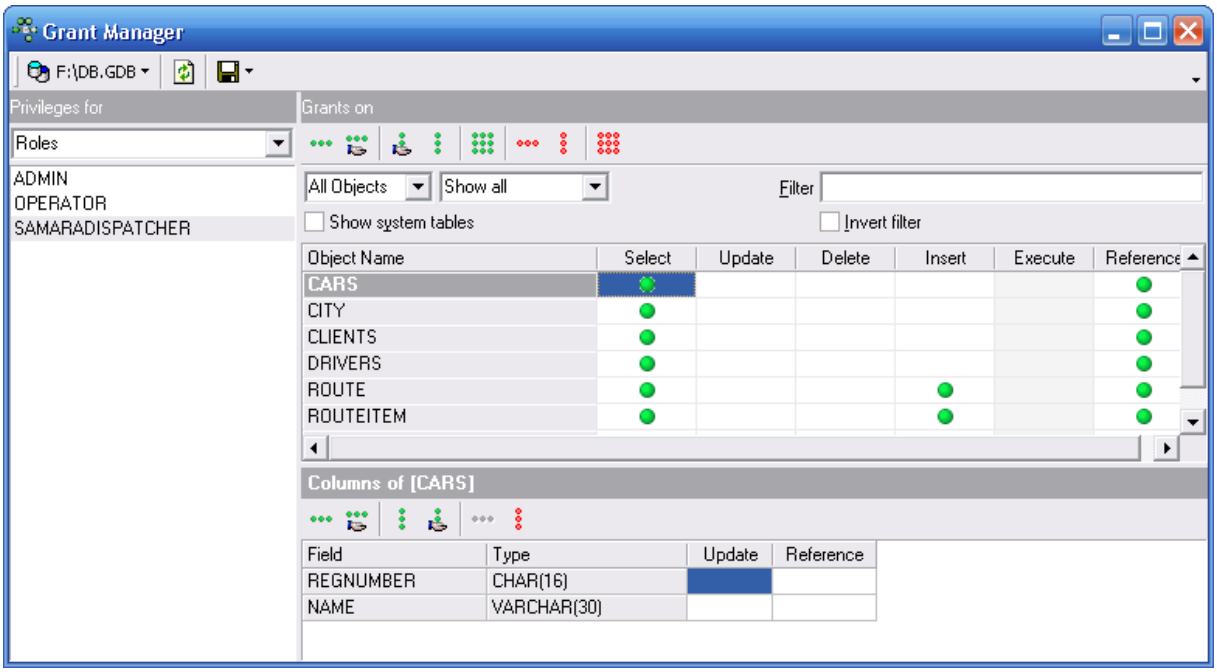
Для задания полномочий можно использовать визуальное средство Grant Manager программы IB Expert. Создание первых двух ролей не потребует практически никакого труда. Так, для первой роли выставим разрешения на все объекты базы данных и все операции с ней (зеленые кружки).



Для второй роли выставим разрешения только на таблицы CARS, CITY, CLIENTS и DRIVERS, а с оставшихся таблиц уберем.



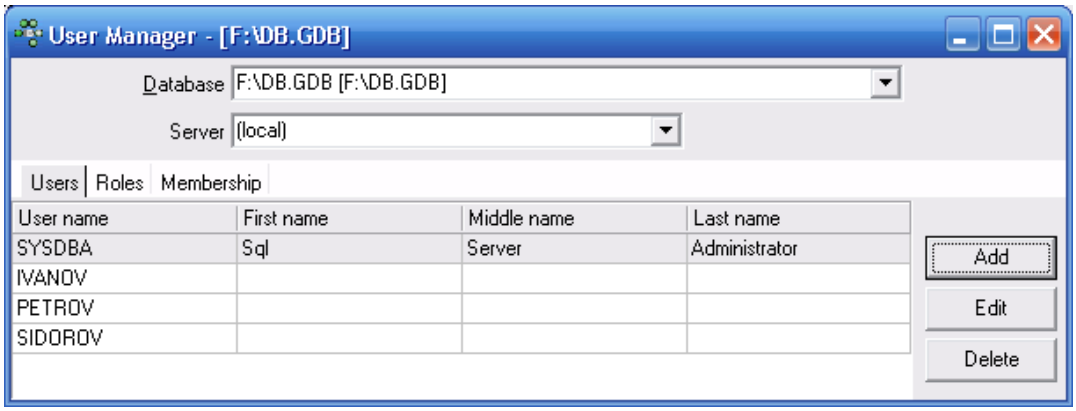
Полностью реализовать функционал для третьей роли с помощью средства Grant Manager не получится. Реализуем только базовый функционал этой роли: разрешим доступ ко всем таблицам базы данных только на чтение кроме двух таблиц: ROUTE и ROUTEITEM, в которые разрешим вставлять новые записи.



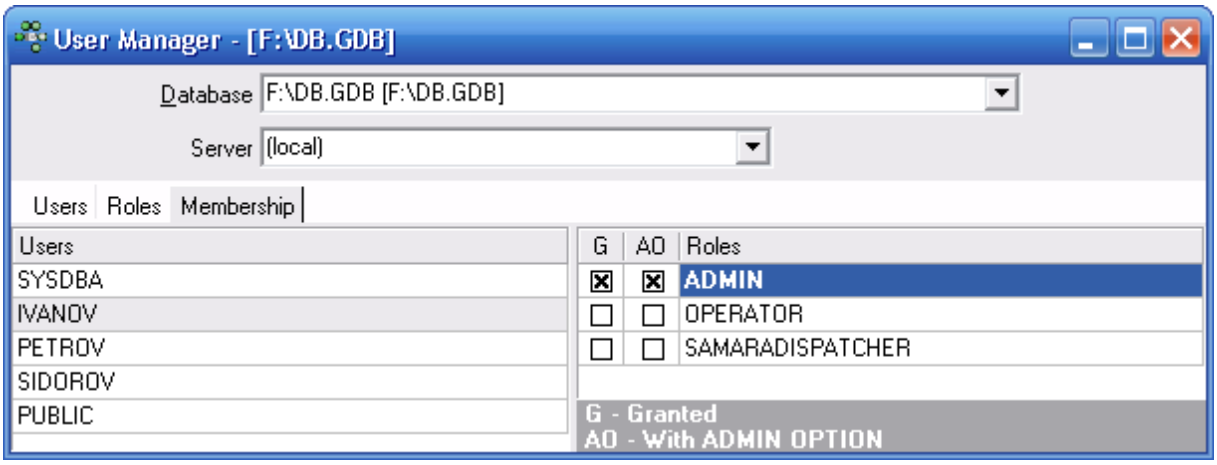
Для того чтобы впоследствии не пришлось производить повторную настройку прав, создадим SQL-скрипт с этих трех ролей:

```
GRANT SELECT, INSERT, UPDATE, DELETE, REFERENCES ON CARS
TO "ADMIN";
GRANT SELECT, INSERT, UPDATE, DELETE, REFERENCES ON CITY
TO "ADMIN";
GRANT SELECT, INSERT, UPDATE, DELETE, REFERENCES ON
CLIENTS TO "ADMIN";
GRANT SELECT, INSERT, UPDATE, DELETE, REFERENCES ON
DRIVERS TO "ADMIN";
GRANT SELECT, INSERT, UPDATE, DELETE, REFERENCES ON ROUTE
TO "ADMIN";
GRANT SELECT, INSERT, UPDATE, DELETE, REFERENCES ON
ROUTEITEM TO "ADMIN";

GRANT SELECT, INSERT, UPDATE, DELETE ON CARS TO OPERATOR;
GRANT SELECT, INSERT, UPDATE, DELETE ON CITY TO OPERATOR;
GRANT SELECT, INSERT, UPDATE, DELETE ON CLIENTS TO
OPERATOR;
GRANT SELECT, INSERT, UPDATE, DELETE ON DRIVERS TO
OPERATOR;

GRANT SELECT, REFERENCES ON CARS TO SAMARADISPATCHER;
GRANT SELECT, REFERENCES ON CITY TO SAMARADISPATCHER;
GRANT SELECT, REFERENCES ON CLIENTS TO SAMARADISPATCHER;
GRANT SELECT, REFERENCES ON DRIVERS TO
SAMARADISPATCHER;
GRANT SELECT, INSERT, REFERENCES ON ROUTE TO
SAMARADISPATCHER;
GRANT SELECT, INSERT, REFERENCES ON ROUTEITEM TO
SAMARADISPATCHER;
```

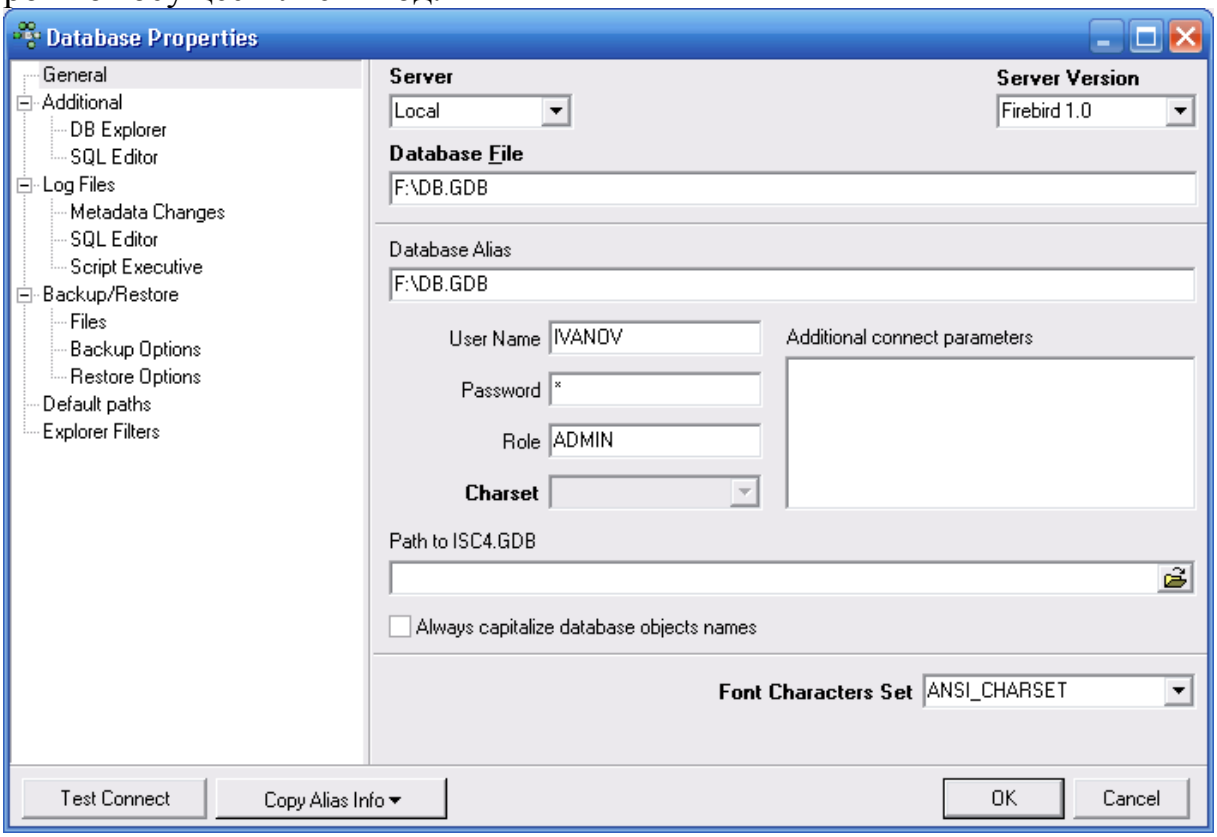
Далее с помощью визуального средства User Manager программы IB Expert создадим трех пользователей: IVANOV, PETROV и SIDOROV



И сопоставим первого пользователя с ролью ADMIN, второго с ролью OPERATOR, а третьего – с ролью SAMARADISPATCHER.



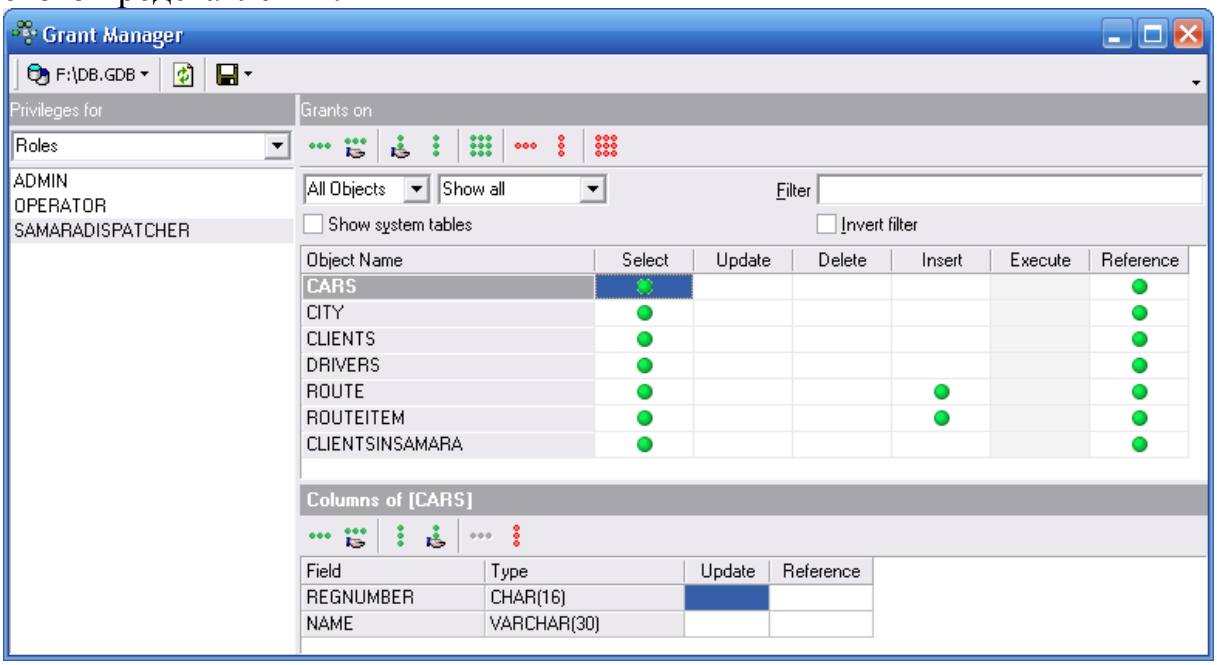
Для того чтобы войти в систему под пользователем IVANOV, необходимо в окне Database Properties указать имя пользователя и роль, под которой тот осуществляет вход.



Теперь приступим к наращиванию функционала роли SAMARADISPATCHER. По условию задачи эта роль должна иметь полномочия выбирать не все записи из таблицы клиентов, а только клиентов, живущих в Самаре. Для этого можно создать представление CLIENTSINSAMARA, которое будет выдавать только соответствующий набор данных.

```
CREATE VIEW CLIENTSINSAMARA ( INN, NAME, ADDRESS, CITYID ) AS
SELECT C.INN, C.NAME, C.ADDRESS, C.CITYID
FROM CLIENTS C
INNER JOIN CITY CI ON C.CITYID=CI.CITYID
WHERE CI.NAME='Самара'
```

Далее разрешаем роли SAMARADISPATCHER получать данные из этого представления:



Добавим в итоговый SQL-скрипт по привилегиям соответствующие команды для нового представления:

```
GRANT SELECT, REFERENCES ON CLIENTSINSAMARA TO SAMARADISPATCHER;
```

Единственную проверку, которую осталось реализовать – возможность роли SAMARADISPATCHER добавлять в маршрут только самарских клиентов. Для этого при вставке новой записи в таблицу ROUTEITEM необходимо проверять город проживания клиента и в случае. Для того, чтобы проверка осуществлялась автоматически можно создать триггер на эту таблицу, который будет это проверять. Однако в этом случае проверка будет осуществляться всякий раз и для любых пользователей. Можно в триггере проверять имя текущего пользователя и только в том случае, если он SIDOROV, то необходимо проверять город проживания клиента. Однако такая настройка существенно усложнит процесс администрирования. Наиболее простой способ настройки с точки зрения последующей настройки представляется создание представления для таблицы ROUTEITEM, которая при вставке записей будет осуществлять данную проверку. Все пользователя работают непосредственно с таблицей ROUTEITEM,

а пользователи, обладающие полномочием SAMARADISPATCHER – через представление ROUTEITEMSAMARA.

Скрипт создания представления: ROUTEITEMSAMARA

```
CREATE VIEW ROUTEITEMSAMARA ( ROUTENUMBER, INN, ORDER_NUMBER ) AS
SELECT RI.ROUTENUMBER, RI.INN, RI.ORDER_NUMBER
FROM ROUTEITEM RI
INNER JOIN CLIENTSINSAMARA C ON RI.INN=C.INN
```

Триггер на вставку записи:

```
CREATE TRIGGER  ROUTEITEMSAMARA_BI FOR
ROUTEITEMSAMARA BEFORE INSERT
AS
  DECLARE VINN VARCHAR(12);
begin
  SELECT INN FROM CLIENTSINSAMARA WHERE INN=NEW.INN INTO VINN;
  INSERT INTO ROUTEITEM (ROUTENUMBER, INN, ORDER_NUMBER) VALUES (NEW.ROUTENUMBER, NEW.INN, NEW.ORDER_NUMBER);
end
```

Единственное, что осталось сделать – настроить доступ к этому представлению только роли SAMARADISPATCHER

8.6. Контрольные вопросы

1. Какой оператор SQL предназначен для выдачи прав?
2. Какой оператор SQL предназначен для отмены прав?
3. Опишите последовательность шагов, которые необходимо сделать, чтобы ограничить доступ какого-либо пользователя к части содержимого какой-либо таблицы.

ЗАКЛЮЧЕНИЕ

В учебном пособии подробно рассмотрен процесс создания баз данных с использованием программных пакетов Power Architect и СУБД Firebird. Применяемый при выполнении лабораторных работ диалект SQL практически полностью совместим со стандартом. Навыки, полученные при выполнении лабораторных работ, позволят без больших усилий перейти на использование любых других реляционных СУБД.

Развитие технологий, связанных с системами управления базами данных не останавливается ни на минуту. Высокий темп разработок диктует сам рынок – для того, чтобы занять достойное место в конкурентной борьбе, фирмам разработчикам баз данных приходится придумывать все новые и новые механизмы в плане надежности хранения данных, обеспечения безопасности, увеличения скорости доступа и т.п.

Только после того, как заинтересованный читатель попробует создать свою собственную базу данных, только после того, как он почувствует искру творчества в этом нелегком деле, только после этого, на взгляд автора, имеет смысл идти дальше в плане повышения уровня своей компетенции в базах данных. Если у кого-то из читателей возникло желание идти вперед в этом направлении, автор считает свою задачу выполненной.

Для более глубокого изучения вопросов реализации баз данных, а также их проектирования, следует обратиться к источникам, указанным в библиографии.

БИБЛИОГРАФИЧЕСКИЙ СПИСОК

1. Коннолли, Томас, Бегг, Каролин. Базы данных. Проектирование, реализация и сопровождение. Теория и практика. 3-е издание / пер. с англ. М.: Издательский дом "Вильямс", 2003. 1440 с.
2. Кренке Д. Теория и практика построения баз данных. 9-е изд. СПб.: Питер, 2005. 859 с.
3. Крутов А.Н. Методы программирования. ООП. UML. RUP: учебное пособие. Самара: Изд-во «Самарский университет», 2004. 116 с.
4. Грабер М. Введение в SQL. М.: ЛОРИ, 1996. 380 с.
5. Том Кайт. Oracle для профессионалов. Книга 2. Расширение возможностей и защита. Второе издание. Киев: ООО “ТИД “ДС”, 2004. 848 с.
6. Хоббс Л., Хилсон С., Лоуенд Ш. Oracle9iR2: разработка и эксплуатация хранилищ баз данных / пер. с англ. М.: КУДИЦ-ОБРАЗ, 2004. 592 с.
7. Ковязин А., Востриков С. Мир InterBase. Архитектура, администрирование и разработка приложений баз данных в InterBase/Firebird/Yaffil. Издание 2-е, дополненное. М.: КУДИЦ-ОБРАЗ, 2002. 496 с.
8. Дж. Грофф, П. Вайнберг. SQL: Полное руководство: пер. с англ. 2-е изд., перераб. и доп. К.: Издательская группа BHV, 2001. 816 с.
9. Карпова Т.С. Базы данных: модели, разработка, реализация: учебник для вузов. СПб.: Питер, 2002. 303 с.

Приложение А

Варианты задач для лабораторной работы 1

1. АИС «Птицефабрика»

Реализация яиц птицефабрикой в ____ месяце

Название породы	План (яиц на 1 птицу)	Кол-во птиц	Получено от реализации	% выполнения плана
До 24 символов	ЦЦЦ	ЦЦЦ	ЦЦЦ.ЦЦ	

Максимальный доход: от реализации яиц у породы _____, на 1 птицу у породы _____

2. АИС «Кадры»

Список работающих в должности _____ на ___/___/_____.

ФИО работающего	Год рождения	Номер цеха	В последней должности (лет)
До 24 символов		ЦЦ	

Время работы в последней должности:
Минимальное _____ лет у _____ (ФИО)
Максимальное _____ лет у _____ (ФИО)

3. АИС «Нормирование труда»

Справка об использовании рабочего времени на дату __/__/____ при выполнении операции _____ рабочими ___ разряда

Цех	ФИО исполнителя	Смена	Затрачено на одну операцию (план)	Затрачено на одну операцию (факт)	Кол-во операций
ЦЦ	До 65 символов	Ц	ЦЦ.Ц	ЦЦЦЦ	

4. АИС «Материально-техническое снабжение»

Ведомость о наличии сырья _____, единица измерения _____, цена _____

Цех	**Поступление сырья**		
	Дата	**Кол-во**	**Стоимость**
До 65 символов		ЦЦЦЦ	ЦЦЦЦ.ЦЦ

5. АИС «Автотранспорт»

Ведомость учета эксплуатационных показателей грузовых машин _____ гаража за _____ месяц текущего года

Марка	**ФИО водителя**	**Пробег**	**Расход горючего**		
			Общий	**На 100 км**	
				Факт	**Норма**
До 12 символов	До 65 символов	ЦЦЦ.Ц	ЦЦЦ.Ц	ЦЦЦ.Ц	ЦЦЦ.Ц

6. АИС «Мороженое»

Справка об отпуске мороженого получателю _____ за _____ месяц текущего года

Наименование	**Упаковка**	**Кол-во упаковок**	**Объем мороженого в упаковке**	**Цена, руб.**	**Стоимость партии**
До 65 символов	До 12 символов	ЦЦЦ	ЦЦ.Ц	ЦЦЦ.ЦЦ	ЦЦЦЦ.ЦЦ

Наибольший объем отпуска _____
Мороженое _____
Стоимость этой партии _____

7. АИС «Гараж»

Свободные места в гаражах города для личных автомобилей марки _____. Дата формирования таблицы _____.

Тип гаража	Наличие мойки	Дата получения информации	Количество свободных мест
До 25 символов	Есть/нет		ЦЦЦ

Максимальное количество мест в гараже номер _____, тел. _____, количество дней с момента получения информации _____.

8. АИС «Вакансия»

График объявления конкурсов на замещение вакантных должностей на _____ семестр текущего учебного года.

Месяц	Кафедра	Должность	Количество мест
	До 25 символов		ЦЦ

Итого вакансий по университету _____

9. АИС «Учет заболеваний»

Справка о выплатах по потере трудоспособности в цехе _____ за _____ месяц

Заболевание	Табельный №	Категория работающего	Процент оплаты	Кол-во дней	Сумма к выплате
До 20 символов	До 12 символов	До 20 символов		ЦЦЦ	ЦЦЦЦ.ЦЦ

Заболевание максимальной длительности у _____ (ФИО)
Дата начала заболевания _____, табельный номер _____

10. АИС «Багаж»

Загрузка багажного отделения _____ на дату _____

Квитанция	Номер места багажа	Ячейка багажного отделения	Характеристика хранения		
			Режим	Кол-во суток	Стоимость
Б/ЦЦЦ	ЦЦ	ЦЦЦ	До 24 символов	ЦЦ	ЦЦЦ.ЦЦ

Максимальное время хранения _____ суток у получателя _____

11. АИС «Планирование учебной нагрузки»

Аудиторная нагрузка кафедры _____ на _____ семестр ____ года

Дисциплина	Поток	Кол-во групп	Кол-во студентов	Кол-во часов		
				Лек.	Лаб.	Упр.
До 24 символов	До 16 символов	ЦЦ	ЦЦЦ	ЦЦ	ЦЦ	ЦЦ

Максимальное время хранения _____ суток у получателя _____

12. АИС «Хлеб»

Возврат хлебобулочных изделий за _____ месяц текущего года из магазина номер _____. Тел: _____, ФИО директора _____

№ накладной	Автофургон	Наименование	Отпуск шт.	Возврат	
				Кол-во	%
ББ.ЦЦЦ	До 32 символов	До 64 символов	ЦЦЦ	ЦЦЦ	

Максимальная сумма возврата _____ руб., дата _____, причина возврата _____

Приложение В

Варианты задач для лабораторной работы 2

1. Автоматизация работы склада готовой продукции.
2. Автоматизация работы пунктов проката кассет.
3. Система учета товаров в магазине.
4. Библиотечная система.
5. Система поддержки составления расписания занятий.
6. Система учета успеваемости студентов на факультете.
7. Система учета успеваемости студентов в ВУЗе.
8. Ведение домашней бухгалтерии.
9. Оценка поставщиков согласно стандарту ГОСТ Р ИСО 9001-2000.
10. Составление кулинарных рецептов.
11. Анализ динамики курсов валют в различных банках.
12. Анализ динамики курсов акций на различных биржах.
13. Автоматизация хранения компакт дисков.
14. Автоматизация процедуры ОСАГО страховым агентом.
15. Ведение истории развития юридических лиц.
16. Ведение штатного расписания для крупной организации.
17. Разработка ежедневника/телефонной книги.
18. АРМ[6].
19. Расчетов за телефонные звонки.
20. АРМ технического обслуживания автомобилей.
21. Опись имущества в крупной организации.
22. Автоматизация документооборота в организации.
23. Учет рабочего времени сотрудников.
24. Создание иерархической записной книжки.
25. Автоматизация процесса планирования проектов.
26. Задача учета компьютеров, включая изменения в их конфигурации.
27. Автоматизация процесса продажи железнодорожных билетов.

[6] АРМ – автоматизированное рабочее место.

Учебное издание

Крутов Алексей Николаевич

БАЗЫ ДАННЫХ

Лабораторный практикум

Публикуется в авторской редакции
Титульное редактирование *С. В. Жидовой*
Компьютерная верстка, макет *Н. П. Бариновой*

Подписано в печать 27.02.13. Формат 60x84/16. Бумага офсетная. Печать оперативная.
Усл.-печ. л. 3,0; уч.-изд. л. 3,25. Гарнитура Times. Тираж 100 экз. Заказ № 2306
Управление по информационно-издательской деятельности Самарского государственного университета: www.infopress.samsu.ru
Издательство «Самарский университет», 443011, г. Самара, ул. Акад. Павлова, 1.
Тел. 8 (846) 334-54-23
Отпечатано на УОП СамГУ