

# **Лабораторные работы по курсу «Базы данных»**

## **Введение**

Данный курс лабораторных работ предназначен для студентов специальностей 010503, 230401, 230105, 090102.

Цель лабораторного практикума – практическое изучение не менее двух СУБД, хотя бы одна из которых должна иметь в качестве языка SQL.

Основные задачи практикума:

- обеспечить студентов конкретным инструментарием для практической реализации методов проектирования БД в курсовой работе;
- изучение современных СУБД

Лабораторный практикум является достаточно самостоятельной частью данного курса.

Предметная область для лабораторных работ должна быть понятной студентам и не требовать специальных знаний. При организации лабораторных работ следует уделять внимание методам реализации в выбранной СУБД следующих задач:

- организация пользовательского интерфейса;
- гибкая система формирования пользователем запросов;
- сетевая и многопользовательская реализация;
- физическая и логическая независимость данных;
- конфигурирование разрабатываемой системы и ведение словаря БД;
- использование генератора отчетов;
- протоколирование проведенных в БД изменений и возможность отката.

При оценке лабораторных работ необходимо учитывать:

- стиль программирования;
- использование современных технологий программирования;
- рациональность принятых решений;
- прозрачность и самодокументируемость программ;
- самостоятельность написания программы.

## Лабораторная работа №1

**Тема:** Основы проектирования структуры БД.

**Цель:** вводная работа, дающая студентам общие представления о моделировании структуры БД, в частности, построении ER-моделей.

**Навыки и умения:** практическое структурирование предметной области, абстрагирование, использование прикладных программных пакетов моделирования структур, использование MS Access для создания БД.

### Теоретический базис

#### Общие понятия

*База данных* – структурированная совокупность данных.

*Система баз данных* - это компьютеризированная система хранения баз данных, основная цель, которой содержать информацию и предоставлять её по требованию.

*Основное назначение СУБД* – обеспечить пользователя инструментом, позволяющим оперировать данными в терминах, не связанных с особенностями их хранения в ЭВМ.

Важнейшим понятием, используемым при проектировании любых информационных систем, является понятие «абстрагирование». *Абстрагирование* – это отбрасывание лишних элементов с выделением основных. Существует несколько уровней абстракции в структурных данных, а именно: функциональный, логический, физический.

С процессом проектирования структуры базы данных связывают следующие уровни абстракции:

- Внешняя модель или уровень представления (описание в терминах пользователей БД);
- Логический или концептуальный уровень (обобщенное описание предметной области, разрабатывается прикладными программистами);
- Внутренний или физический уровень (описание концептуальной модели на языке некоторой СУБД).

Так как на текущий момент наиболее распространенными являются реляционные БД, то упор в курсе сделан именно на БД этого класса.

В качестве концептуальной модели выступает ER-модель.

ER-модель (с англ. «сущность/связь») позволяет достаточно легко описать условия целостности, но вместе с тем, для этих моделей сложно проводить формальную оптимизацию.

#### Основные понятия реляционной модели

В соответствии с реляционной моделью, БД является совокупностью отношений.

*Отношение* – это некоторое подмножество прямого произведения. В качестве альтернативного определения применимо: *отношение* – это множество кортежей. Размер кортежа называют арностью отношения.

На практике рассматривается только конечное множество кортежей. Естественно конечное множество кортежей записывается в виде таблиц, т.е. таблица – это *отношение*.

Столбцу таблицы соответствует некоторый атрибут (объекта). Значение этого атрибута выбирается из некоторого множества, которое называется *доменом*. Строку таблицы образует некоторое упорядоченное множество значений всех атрибутов таблицы, взятых из своих доменов.

Альтернативное название атрибутов – *поле*. На физическом уровне строка таблицы (некоторый кортеж) называется *хранимой записью*. Хранимая запись, состоит из хранимых полей (значение атрибутов, взятых из своих доменов). Совокупность хранимых записей (таблица) называется *хранимый файл* (хранимая БД).

Атрибуты или множество атрибутов значения, которых уникальным образом идентифицируют экземпляр объекта, называются *первичным ключом*, т.к. все экземпляры объекта должны быть различны, то каждый объект должен иметь ключ.

*Потенциальный ключ* – это обобщение понятия первичного ключа. Потенциальные ключи также как и первичный обладают свойством уникальной идентификации кортежа в отношении, но если первичный ключ в отношении должен быть выбран только один, то потенциальных ключей может быть несколько (первичный ключ выбирается из потенциальных).

*Внешний ключ* – это множество атрибутов объекта; каждому значению внешнего ключа соответствует значение потенциального ключа. Внешние ключи используются для связывания кортежей в реляционных базах данных.

### **Общий подход к ER-моделированию**

Для того, чтобы представить как устроена предметная область нужно задать множество объектов реального мира (главная проблема что считать объектом). Объект – семантическое понятие, которое может быть полезно при обсуждении устройств реального мира. Сущность реального мира – объекты – не обязательно материальны – важно понятие существенно и различимо для других.

Пример: объектами являются: студент, человек, преподаватель, аудитория  
Различают *тип* объекта и *экземпляр* объекта

Пример: тип – аудитория; экземпляр – 358.

Объект обладает рядом свойств, которые иногда называют *атрибутами* объекта (набор атрибутов одноименных объектов в различных прикладных задачах может различаться).




Пример: человек обладает атрибутами фамилия, имя, отчество, дата рождения.

Для определения нового объекта иногда используют понятие *подтипа* (пилот – есть подтип сотрудник, обладает всеми свойствами сотрудника и, кроме того, свойствами: список разрешенных самолетов).

Между объектами могут возникать связи трех видов:

- один к одному 1:1 (пациент: место в палате);
- один к многим 1:n и многие к одному n:1;
- многие ко многим n:n (пациент : хирург).

При построении моделей используются следующие геометрические фигуры:

Объект -  Связь-  Атрибут - 

В настоящее время существует большое множество прикладных программ для создания графического представления структуры БД. При этом могут быть использованы как специализированные средства (например, Visio), так и средства построения графических образов (Umbrello, OO Draw).

## **Основы работы с MS Access**

### **Создание новой БД в среде MS Access**

MS Access является частью пакета MS Office и представляет собой СУБД. Запустите MS Access, используя команды меню «Пуск» -> «Программы» -> «Microsoft Office» -> «Access». Перед вами откроется основное окно программы. Выполните пункты меню «Файл» - «Создать», после чего из вариантов выберите «Новая база данных» и сохраните вашу БД на диск.

Открывшееся окно представляет собой новую БД.

### **Объекты БД в среде MS Access**

MS Access поддерживает следующие объекты:

- Таблица – отношение в реляционной терминологии;
- Запрос – сохраненный текст (модель) запроса на языке SQL к таблицам БД;
- Формы – конструирование интерфейса пользователя;
- Отчеты – конструирование выходной информации БД;
- Страницы – страницы доступа через web;
- Макросы – группы макрокоманд;
- Модули – программные модули на языке VBA, которые могут быть использованы в запросах, формах, отчетах, макросах.

### **Создание таблиц в среде MS Access**

СУБД MS Access поддерживает различные варианты создания таблиц. Рассмотрим наиболее часто используемый вариант создания – в режиме конструктора. Для этого среди объектов БД выберите «Таблицы», после чего выполните пункт «Создание таблицы в режиме конструктора».

В появившемся окне предлагается ввести имена атрибутов таблицы с указанием их типов. Типы атрибутов выбираются из выпадающего списка вариантов, а конкретные характеристики указываются на панели внизу (например, для текстового формата можно указать максимальный размер). Третий столбец на форме конструирования отношений – описание атрибутов. При создании сложных БД с большим числом отношений и атрибутов в них обязательно описывайте логику, которую вы закладываете в тот или иной атрибут.

После создания всех атрибутов не забудьте указать ключевое поле вашего отношения. Для этого выделите нужные поля и нажмите на иконку «ключевое поле» изображающую ключ на панели конструктора.

### Создание схем данных в среде MS Access

СУБД MS Access реализует инструмент проектирования связей между таблицами (объектами) БД. Этот инструмент называется «Схема данных».

Для вызова этого инструмента можете воспользоваться соответствующей кнопкой на панели «База данных» (рис. 1.1)

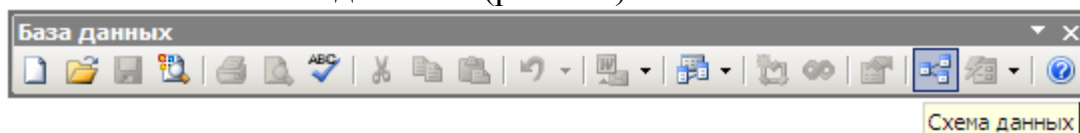


Рисунок 1.1 – Панель «База данных»

В рабочую область схемы можно добавлять и удалять таблицы, а также связи между ними. Для добавления связи между таблицами выберите необходимое поле (внешний ключ) и используя Drag'n'Drop протяните его до нужного поля связанной таблицы (потенциальный ключ). После чего, в появившемся окне свойств вновь созданной связи, можете выставить необходимые настройки (рис. 1.2).

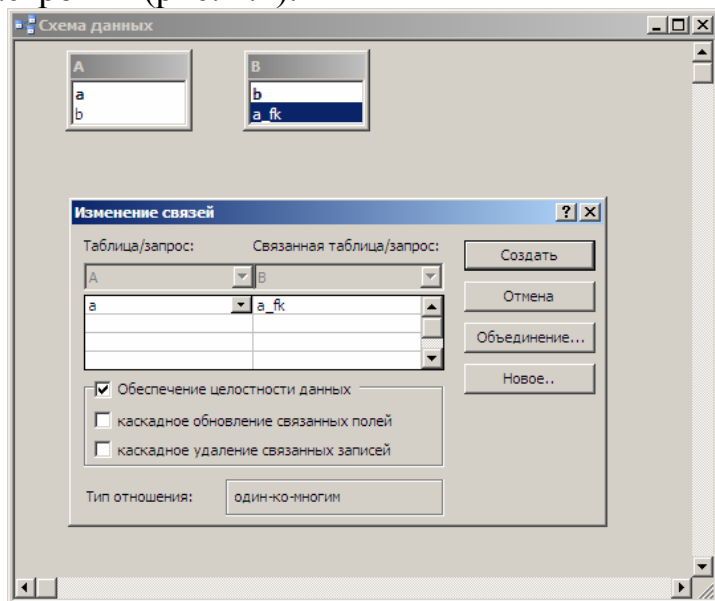


Рисунок 1.2 – Редактирование связей в MS Access

После окончания редактирования не забудьте сохранить схему.

# **Задание на лабораторную работу №1**

## **Часть I**

Выбрать предметную область (например: склад, больница, аптека, аэропорт и т.д.) и составить для нее:

- а) описание предметной области (от имени конечного пользователя);
- б) ER-диаграмму.

Ограничение: от 5ти до 10ти сущностей для описания области. Каждый объект должен иметь хотя бы один атрибут.

Описание и диаграмма включается в отчет по лабораторной работе. Отчет выполняется в печатном виде на листах формата А4 согласно общепринятым правилам оформления лабораторных работ.

## **Часть II**

Используя MS Access перенести полученную модель в БД, используя таблицы и схему данных.

## **Прием работы**

Прием происходит при наличии оформленного отчета и работающей БД, созданной в среде MS Access.

## **Вопросы**

1. Что такое база данных?
2. Что такое система баз данных?
3. Что такое система управления базами данных?
4. Основное назначение?
5. Основные компоненты СУБД?
6. Что подразумевает понятие абстрагирование в СУБД?
7. Какие существуют уровни абстракции в структурных данных?
8. Опишите уровень представления
9. Опишите концептуальный уровень
10. Опишите физический уровень
11. Виды связей
12. Что такое отношение (таблица) в реляционной модели СУБД?
13. Что такое домен в реляционной модели СУБД?
14. Что такое атрибут (поле) в реляционной модели СУБД?
15. Что такое картеж (хранямая запись) в реляционной модели СУБД?
16. Что такое первичный ключ?
17. Что такое потенциальный ключ?
18. Что такое внешний ключ?

## Лабораторная работа №2

**Тема:** Выполнение операций над данными с использованием операторов языка SQL.

**Цель:** научиться использовать операторы языка SQL для работы с данными БД.

**Навыки и умения:** составление и выполнение SQL-запросов в среде MS Access.

### Теоретический базис

#### Описание языка

Язык SQL – стандартный язык запросов по работе с реляционными базами данных. Язык SQL появился после реляционной алгебры, и его прототип был разработан в конце 70-х годов в компании IBM Research. В силу своего широкого распространения постепенно стал стандартом «де-факто» для языков манипулирования данными в реляционной СУБД.

Язык SQL предназначен для выполнения операций над таблицами (создание, удаление, изменение структуры) и над данными таблиц (выборка, изменение, добавление и удаление), а также некоторых сопутствующих операций. SQL является непроцедурным языком и не содержит операторов управления, организации подпрограмм, ввода-вывода и т.п. В связи с этим SQL автономно не используется, обычно он погружен в среду встроенного языка программирования СУБД.

Основным назначением языка SQL (как и других языков для работы с базами данных) является подготовка и выполнение запросов. В результате выборки данных из одной или нескольких таблиц может быть получено множество записей, называемое представлением.

Представление, по существу является таблицей, формируемой в результате выполнения запроса. Оно является разновидностью хранимого запроса. По одним и тем же таблицам можно построить несколько представлений.

#### Оператор SELECT

Язык запросов в SQL состоит из единственного оператора – SELECT. Синтаксис оператора SELECT имеет следующий вид:

```
SELECT [ ALL| DISTINCT] <Список полей>|*  
FROM <Список таблиц>  
[WHERE <Предикат-условие выборки или соединения>]  
[GROUP BY <Список полей результата>]
```

*[HAVING <Предикат-условие для группы>]*

*[ORDER BY <Список полей, по которым упорядочить вывод>];*

**SELECT** – ключевое слово, которое сообщает СУБД, что эта команда – запрос. Все запросы начинаются этим словом с последующим пробелом. За ним может следовать способ выборки.

Здесь ключевое слово **ALL** означает, что в результирующий набор строк включаются все строки, удовлетворяющие условиям запроса. Значит в результирующий набор могут попасть одинаковые строки. Это нарушение принципов теории отношений (в отличие от реляционной алгебры, где по умолчанию предполагается отсутствие дубликатов в каждом результирующем отношении).

Ключевое слово **DISTINCT** означает, что в результирующий набор включаются только различные строки, то есть дубликаты строк результата не включаются в набор.

Список полей – это список перечисленных через запятую столбцов, которые выбираются запросом из таблиц.

Символ \* (звездочка) означает, что в результирующий набор включаются все столбцы из исходных таблиц запроса.

В разделе **FROM** задается перечень исходных отношений (таблиц) запроса. В случае, если указано более одного имени таблицы, неявно подразумевается, что над перечисленными таблицами осуществляется операция декартова произведения.

Разделы **SELECT** и **FROM** являются обязательными, все другие разделы являются необязательными.

*Примечание: далее в рамках будут приводиться примеры SQL-запросов. Для улучшения восприятия материала и понимания технологии написания SQL-запросов студентам рекомендуется воссоздать описанную схему таблиц и попробовать все из рассмотренных в примерах запросы прежде чем приступать к выполнению лабораторной работы.*

Допустимо в раздел **SELECT** включать не только имена полей и \*, но также и функции языка SQL (стандартные, либо описанные в СУБД). Более подробную информацию о функциях вы можете получить в справке к конкретной СУБД. Кратко рассмотрим несколько из функций, доступных в MS Access:

**Ucase(выражение)** – переводит значения поля «выражение» в верхний регистр;

**Mid(строка, начало\_поиска[, длина])** – возвращает строку из поля «строка», начиная с позиции «начало\_поиска» длиной – «длина». Если длина не указана, то до конца строки.

**Round(выражение [,количество\_десятичных \_знаков])** – округляет «выражение» с точностью «количество\_десятичных\_знаков». Если второй аргумент не указан, то округляет до целого.

**IFF(выражение, truepart, falsepart)** – возвращает одно из значений: truepart, если выражение истинно и falsepart – если ложно.



### Пример 2.1:

Рассмотрим базу данных, которая моделирует сдачу сессии в некотором учебном заведении. Пусть она состоит из трех отношений R1, R2, R3, представленных таблицами R1, R2, R3 соответственно.

*Примечание: В дальнейшем эти отношения будут использоваться для всех примеров в данной лабораторной работе.*

R1 = (ФИО, Дисциплина, Оценка); R2 = (ФИО, Группа); R3 = (Группа, Дисциплина).

R1		
ФИО	Дисциплина	Оценка
Петров Ф.И.	Базы данных	5
Сидоров К.А.	Базы данных	4
Миронов А.В.	Базы данных	2
Петров Ф.И.	Моделирование	5
Сидоров К.А.	Моделирование	4
Миронов А.В.	Моделирование	Null
Трофимов П.А.	Сети ЭВМ	4
Иванова Е.А.	Сети ЭВМ	5
Уткина Н.В.	Сети ЭВМ	5

R2	
ФИО	Группа
Петров Ф.И.	АИ21
Сидоров К.А.	АИ21
Миронов А.В.	АИ21
Трофимов П.А.	АИ22
Иванова Е.А.	АИ22
Уткина Н.В.	АИ22

R3	
Группа	Дисциплина
АИ21	Базы данных
АИ21	Моделирование
АИ22	Сети ЭВМ

Тогда запрос всех студентов и дисциплин, по которым у них есть оценки будет выглядеть следующим образом:

***SELECT ФИО, Дисциплина FROM R1;***

В разделе **WHERE** задаются условия отбора строк результата или условия соединения кортежей исходных таблиц, подобно операции условного соединения в реляционной алгебре.

В выражении условий раздела **WHERE** могут быть использованы следующие предикаты:

**Предикаты сравнения** (=, <>, >, >=, <, <=), которые имеют традиционный смысл.

**Предикат Between A and B** – принимает значения между A и B. Предикат истинен, когда сравниваемое значение попадает в заданный диапазон, включая границы диапазона. Одновременно в стандарте задан и противоположный предикат Not Between A and B, который истинен тогда,

когда сравниваемое значение не попадает в заданный интервал, включая его границы.

**Предикат вхождения в множество IN** (множество) истинен тогда, когда сравниваемое значение входит в множество заданных значений. При этом множество значений может быть задано простым перечислением или встроенным подзапросом. Одновременно существует противоположный предикат **NOT IN** (множество), который истинен тогда, когда сравниваемое значение не входит в заданное множество.

**Предикаты сравнения с образцом LIKE и NOT LIKE.** Предикат **LIKE** требует задания шаблона, с которым сравнивается заданное значение, предикат истинен, если сравниваемое значение соответствует шаблону, и ложен в противном случае. Предикат **NOT LIKE** имеет противоположный смысл. Шаблон может содержать % (\* для Access) для обозначения любого числа любых символов; \_ (? для Access) для обозначения любого одного символа.

**Предикат сравнения с неопределенным значением IS NULL.** Для выявления равенства значения некоторого атрибута неопределенному значению применяют специальные стандартные предикаты: <имя атрибута> **IS NULL** и <имя атрибута> **IS NOT NULL**

**Предикат существования EXIST и не существования NOT EXIST.** Применяется во вложенных запросах для определения непустого или пустого множества, являющегося результатом выборки.

В условиях поиска могут быть использованы все рассмотренные предикаты.

**Пример 2.2:**

Например, можно выбрать из отношения R3 только те дисциплины, которые были у группы АИ21:

***SELECT Дисциплина FROM R3 WHERE Группа Like 'АИ21';***

Для таблиц и полей можно задавать псевдонимы (alias). Для этого необходимо использовать предлог **AS**. Например, **Select [Цена за единицу] \* [Количество] as [Стоимость покупки] from Продажа;** - здесь определяется псевдоним для вычисляемого поля (операция умножение).

В разделе **GROUP BY** задается список полей группировки. **GROUP BY** группирует записи данных и объединяет в одну запись все записи данных, которые содержат идентичные значения в указанном поле (или полях). **WHERE** определяет, какие записи должны участвовать в группировании, т.е. фильтрует до группирования.

Обратите внимание, что использование **Group By** отличается от использования **Distinct**. Во втором случае будут отброшены кортежи, которые в текущем представлении совпадают по всем полям (из совпадающих записей остается только один кортеж). Операция группировки приводит исходное отношение к виду, когда ко всем полям, запрошенным на отображение и не указанным в выражении группировки, применяются агрегатные функции (если они не определены, то запрос не выполнится).

### Пример 2.3:

Для того чтобы почувствовать разницу между использованием ключевого слова **DISTINCT** и группировкой с помощью **GROUP BY** попробуйте поочередно выполнить следующие запросы к отношению R2:

```
SELECT DISTINCT ФИО, Группа FROM R2;  
SELECT Группа FROM R2;  
SELECT DISTINCT Группа FROM R2;  
SELECT max(Поле1), Поле2 FROM R2 Group by Поле2;  
SELECT Поле2 FROM R2 Group by Поле2;
```

В разделе **HAVING** задаются предикаты-условия, накладываемые на каждую группу. **HAVING** используется для фильтрации записей, полученных в результате группировки. **WHERE** определяет, какие записи должны участвовать в группировании, т.е. фильтрует до группирования. **HAVING** определяет, какие из получившихся в результате группировки записей будут включены в результирующую выборку, т.е. фильтрует записи после группирования.

### Пример 2.4:

Для того чтобы наложить фильтр на поле в запрос, данные в котором уже были сгруппированы, необходимо использовать **HAVING**:

```
SELECT R2. Группа FROM R2  
GROUP BY R2. Группа  
HAVING R2. Группа = 'АИ21';
```

В части **ORDER BY** задается список полей упорядочения результата, то есть список полей, который определяет порядок сортировки в результирующем отношении. Например, если первым полем списка будет указан Шифр группы, а вторым Фамилия, то в результирующем отношении записи сначала будут расположены в порядке возрастания шифра группы, а затем в рамках одной группы записи будут отсортированы по фамилии в алфавитном порядке.

## Применение агрегатных функций

В SQL добавлены дополнительные функции, которые позволяют вычислять обобщенные групповые значения. Для применения агрегатных функций предполагается предварительная операция группировки. При группировке все множество кортежей отношения разбивается на группы, в которых объединяются кортежи, имеющие одинаковые значения атрибутов, которые заданы в списке группировки.

### Пример 2.5:

Для того, чтобы посчитать, сколько студентов обучается в группе АИ22 необходимо выполнить запрос:

```
SELECT COUNT(*), Группа FROM R2  
WHERE Группа Like 'АИ21';
```

Некоторые агрегатные функции описаны в таблице 1.

Таблица 1 - Агрегатные функции

Функция	Результат
COUNT	Количество строк или непустых значений полей, которые выбрал запрос
SUM	Сумма всех выбранных значений данного поля
AVG	Среднеарифметическое значение всех выбранных значений данного поля
MIN	Наименьшее из всех выбранных значений данного поля
MAX	Наибольшее из всех выбранных значений данного поля

Агрегатные функции применяются подобно именам полей в операторе SELECT, но они используют имя поля как аргумент. С функциями SUM И AVG могут использоваться только числовые поля. С функциями COUNT, MAX, MIN могут использоваться как числовые, так и символьные поля. При использовании с символьными полями MAX и MIN будут транслировать их в эквивалент ASCII кода и обрабатывать в алфавитном порядке.

### Применение объединения (JOIN)

Стандарт SQL2 расширил понятие условного объединения. В стандарте SQL1 при объединении отношений использовались только условия, задаваемые в части WHERE оператора SELECT, и в этом случае в результирующее отношение попадали только сцепленные по заданным условиям кортежи исходных отношений, для которых эти условия были определены и истинны. Однако в действительности часто необходимо объединять таблицы таким образом, чтобы в результат попали все строки из первой таблицы, а вместо тех строк второй таблицы, для которых не выполнено условие соединения, в результат попадали бы неопределенные значения. Или наоборот, включаются все строки из правой (второй) таблицы, а отсутствующие части строк из первой таблицы дополняются неопределенными значениями. Такие объединения были названы внешними в противоположность объединениям, определенным стандартом SQL1, которые стали называться внутренними.

**Внутреннее объединение (INNER JOIN)** возвращает записи из двух таблиц, если значение первичного ключа первой таблицы соответствует значению внешнего ключа второй таблицы, связанной с первой.

Формат описания:

*<выражение естественного объединения> -*

*<имя таблицы1> NATURAL { INNER | FULL [OUTER]  
LEFT [OUTER] | RIGHT [OUTER] } JOIN <имя таблицы2>|*

*<выражение объединения> -*

*<имя таблицы1> { INNER | FULL [OUTER] | LEFT [OUTER] | RIGHT [OUTER] }  
JOIN { ON условие | [USING  
(список столбцов)] } <имя таблицы2>*

*<выражение перекрестного объединения> -*

*<имя таблицы1> CROSS JOIN <имя таблицы2>*

*<выражение запроса на объединение> -  
<имя таблицы1> UNION JOIN <имя таблицы2>*

В этих определениях **INNER** – означает внутреннее объединение, **LEFT** – левое объединение, то есть в результат входят все строки первой таблицы, а части результирующих кортежей, для которых не было соответствующих значений во второй таблице, дополняются значениями **NULL** (не определено). Ключевое слово **RIGHT** означает правое внешнее соединение, и в отличие от левого соединения в этом случае в результирующее отношение включаются все строки второй таблицы, а недостающие части из первой таблицы дополняются неопределенными значениями. Ключевое слово **FULL** определяет полное внешнее объединение: левое и правое. При полном внешнем объединении выполняются и правое и левое внешние объединения и в результирующее отношение включаются все строки из первой таблицы, дополненные неопределенными значениями, и все строки из второй таблицы, также дополненные неопределенными значениями. Ключевое слово **OUTER** означает внешнее объединение, но если заданы ключевые слова **FULL**, **LEFT**, **RIGHT**, то объединение всегда считается внешним.

#### **Пример 2.6:**

Два следующих запроса выведут один и тот же результат:

```
SELECT R3.*,R2.* FROM R2,R3  
WHERE R2.Группа=R3.Группа;  
SELECT R3.*,R2.* FROM R2 INNER JOIN R3  
ON R2.Группа=R3.Группа;
```

### **Перекрестные запросы и подзапросы**

Перекрестный запрос – способ группировки данных по двум измерениям, позволяющий отображать итоги в компактном результирующем наборе. В перекрестном запросе группировка выполняется по одному полю, а итоговая функция применяется к другому полю. Структура перекрестного запроса следующая: в конструкции **TRANSFORM** указывается поле и групповая функция, применяемая к нему, данное поле выводится на пересечении строк и столбцов; в конструкции **GROUP BY** указывается поле, по которому проводится группировка и которое выводится в качестве заголовков строк; в конструкции **PIVOT** указывается поле, значения которого выводятся в качестве заголовков столбцов.

#### **Пример 2.7.**

Вычислить средние оценки по каждой дисциплине в каждой группе.

```
TRANSFORM Avg(R1.Оценка) AS [СредняяОценка]  
SELECT R3.Дисциплина FROM (R2 INNER JOIN R1 ON R2.ФИО =  
R1.ФИО) INNER JOIN R3 ON R2.Группа = R3.Группа  
GROUP BY R3.Дисциплина PIVOT R2.Группа;
```

## Основы работы с MS Access

### Создание SQL-запросов в среде MS Access

Для создания запроса необходимо перейти на вкладку «запросы» в вашей БД и выполнить «Создание запроса в режиме конструктора».

В появившемся диалоге добавления таблиц в конструкторе запросов нажать «Заккрыть», а затем на панели «Конструктор запросов» выбрать «Вид» запроса – SQL (рис. 2.1). Переключаться между видом конструирования запроса можно, используя контекстное меню.

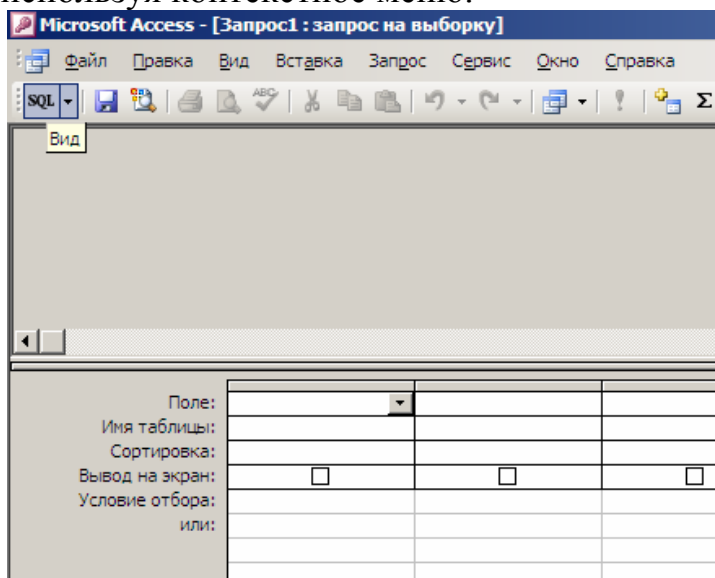


Рисунок 2.1 – Переключение в режим SQL

После выполнения этих действий вам будет предложено окно для ввода текста запроса. По окончании ввода можете выполнить запрос для просмотра результатов, используя кнопку на панели «Конструктор запросов» (рис. 2.2).

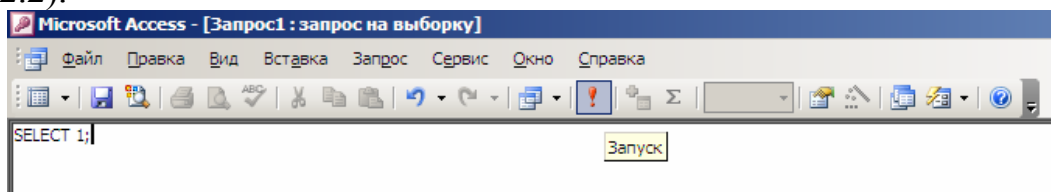


Рисунок 2.2 – Запуск запроса

Запросу можно передать параметр. Для этого надо в тексте запроса написать [param] (вместо param может быть любое слово записанное на кириллице или латинице, кроме служебных слов SQL и имен полей используемых отношений) – при выполнении запроса будет запрошен ввод параметра. Несколько параметров с одинаковым именем считаются одним.

## Использование базы данных авто

### Описание БД

Для выполнения лабораторных работ необходимо использовать базу данных автомобилей (файл allauto.mdb). Данная БД состоит из двух таблиц:

AUTO – содержит сведения о автомобилях;

MENU – описывает свойства автомобилей.

## Описание полей

Ниже приведена таблица с описанием необходимых для выполнения лабораторной работы полей таблицы AUTO (название, тип, длина и описание):

Field	Field Name	Type	Width	Dec
1	INV	Numeric	5	ИНВЕНТАРНЫЙ НОМЕР АВТ.
2	C_TYPE	Character	2	КОД ТИПА
3	C_PLANT	Character	2	КОД ЗАВОДА-ИЗГОТОВИТЕЛЯ
4	C_MARK	Character	2	КОД МАРКИ (МОДЕЛИ)
5	C_BODY	Character	2	КОД ТИПА КУЗОВА
6	C_GROUP	Character	2	КОД ШТАТНОЙ ГРУППЫ
7	N_CHASS	Character	8	НОМЕР ШАССИ
8	N_ENG	Character	8	НОМЕР ДВИГАТЕЛЯ
9	N_BODY	Character	8	НОМЕР КУЗОВА
10	YEAR	Numeric	4	ГОД ВЫПУСКА
11	N_PASS	Character	10	НОМЕР ТЕХ.ПАСПОРТА
12	D_PASS	Date	8	ДАТА ТЕХ.ПАСПОРТА
13	SIGN	Character	7	ОСНОВ.НОМЕР ГОС.РЕГИСТРАЦИИ
14	SIGN1	Character	7	1 ДОП.НОМЕР ГОС.РЕГИСТРАЦИИ
15	SIGN2	Character	7	2 ДОП.НОМЕР ГОС.РЕГИСТРАЦИИ
16	D_SIGN	Date	8	ДАТА ВЫДАЧИ НОМ.ГОС.РЕГИСТР.
17	D_EXPL	Date	8	ДАТА ВВОДА В ЭКСПЛУАТАЦИЮ
18	C_CLASS	Character	2	КОД КЛАССА
19	C_COLOR	Character	2	КОД ЦВЕТА
20	C_DUTY	Character	2	КОД СЛУЖБЫ ЭКСПЛУАТАЦИИ
21	C_OWNER	Character	4	КОД ВЛАДЕЛЬЦА
22	COST	Numeric	8	СТОИМОСТЬ
26	C_SOUR	Character	2	КОД ИСТОЧНИКА ПОЛУЧЕНИЯ
27	N_SOUR	Character	15	НОМ.фондового извещения
28	D_SOUR	Date	8	ДАТА фондового извещения
29	C_STOR	Character	2	КОД СКЛАДА
30	N_STOR	Character	15	НОМЕР ДОК.СКЛАДА
31	D_STOR	Date	8	ДАТА ПОЛУЧЕНИЯ СО СКЛАДА
32	N_ALOC	Character	15	НОМЕР ДОК. НА РАСПРЕДЕЛ. В ПОДР.
33	D_ALOC	Date	8	ДАТА ДОК. НА РАСПРЕДЕЛ. В ПОДР.
34	D_TRANS	Date	8	ДАТА ПЕРЕДАЧИ В ПОДРАЗДЕЛЕНИЕ
35	D_DEL	Date	8	ДАТА СПИСАНИЯ (ЕСЛИ СПИСАН)

## **Задание на лабораторную работу №2**

### **Часть I Простые запросы с параметром**

1. Написать запрос для выбора автомобилей определенного цвета. Цвет задается в виде параметра в условии WHERE (например, 'белый').

### **Часть II Использование агрегирования и подзапросов**

1. Определить количество автомобилей, у которых номер фондового извещения начинается на "10" и не заканчивается на "39"
2. По каждой штатной группе а/м определить, сколько а/м каждой марки было выпущено в заданном году. Вывести названия групп и названия марок на экран.
3. Определить, какие а/м данного класса переданы в подразделения после указанной даты. Указать также номер автомобиля и дату документа передачи каждого а/м.

### **Часть III Использование объединений**

1. Произвести выборку автомобилей из двух полей «номер авто», «класс авто» (подставлять название из отношения MENU). Если поле «класс» в таблице MENU не существует, то выводить строку «Класс средства неизвестен» с помощью функции iif.

### **Часть IV Использование перекрестных запросов и подзапросов**

1. Определить, сколько а/м каждой марки имеют год выпуска меньший, чем округленный до целого средний год выпуска а/м заданной пользователем марки.
2. Определить какое количество а/м каждой марки в каком году было произведено (перекрестный запрос: марки а/м на год производства).

### **Прием работы**

Прием происходит при наличии оформленного отчета и работающей БД, созданной в среде MS Access.

### **Вопросы**

1. Что такое SQL, назначение языка SQL?
2. Назначение команды SELECT?
3. Что такое внешнее и внутреннее объединение, чем отличаются?
4. Что такое левое, правое и полное объединение?
5. Что такое перекрестный запрос?
6. Как применить агрегатную функцию?
7. Для чего в стандарт SQL2 были введены объединения?
8. Чем отличается использование WHERE от HAVING?
9. Чем отличается использование DISTINCT от группировки?



## Лабораторная работа №3

**Тема:** Изменение данных и структуры БД. Клиентский интерфейс для БД. Многопользовательские БД.

**Цель:** развитие у студентов навыков программирования приложений, использующих БД, знакомство с частями SDL и DML языка SQL.

**Навыки и умения:** модификация данных и определение структуры БД с помощью SQL, использование инструментария MS Access (редактор макросов, VBA модули, конструктор форм), написание клиентского интерфейса, работа с многопользовательскими БД.

### Теоретический базис

#### Многопользовательские БД

Работа на изолированном компьютере с небольшой базой данных в настоящий момент становится нехарактерной для большинства приложений. БД отражает информационную модель реальной предметной области, хранит большие объемы информации, которая постоянно увеличивается. Соответственно увеличивается количество приложений, работающих с единой базой данных. Компьютеры объединяются в локальные сети и осуществляют доступ к корпоративной базе данных общего пользования, расположенной на сервере.

Существует два варианта организации базы данных в локальной сети.

Первый вариант – **системы распределенной обработки данных**. БД расположена на одной машине (сервере). К ней осуществляется параллельный доступ нескольких пользователей.

Второй вариант – **системы распределенных баз данных**. БД распределена на нескольких компьютерах, объединенных в сеть. К БД возможен параллельный доступ нескольких пользователей. Это режим параллельного доступа к распределенной БД.

В общем случае режимы использования БД можно представить в следующем виде (рис. 3.1).

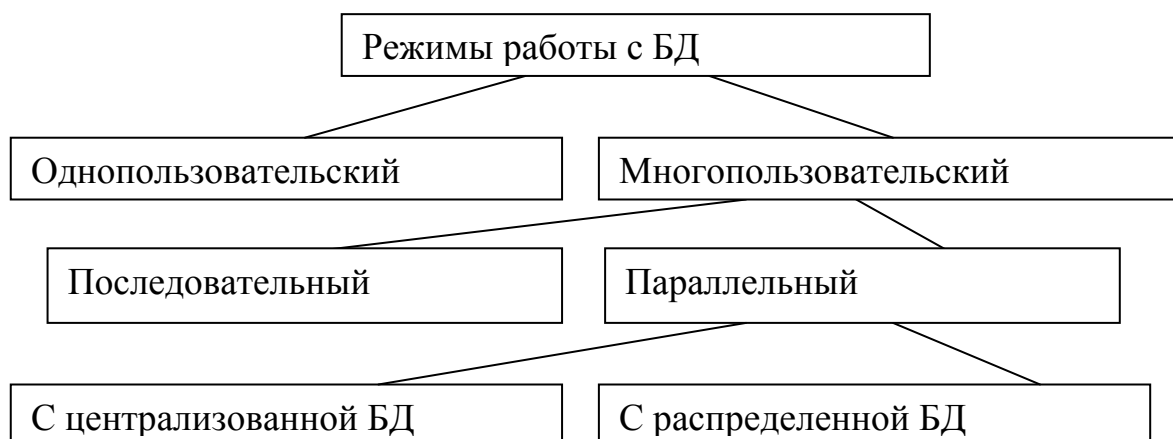


Рис. 3.1 Режимы работы с базой данных

Для организации коллективного доступа в СУБД применяется **механизм блокировок**. Суть блокировки состоит в том, что на время выполнения какой-либо операции в БД доступ к используемому объекту со стороны других потребителей временно запрещается или ограничивается. Например, при копировании таблицы она блокируется от изменения, хотя и разрешено просматривать ее содержимое.

Выделим четыре вида блокировок, перечисленных в порядке убывания строгости ограничений на возможные действия:

- полная блокировка;
- блокировка от записи;
- предохраняющая блокировка от записи;
- предохраняющая полная блокировка.

**Полная блокировка.** Означает полное завершение всяких операций над основными объектами (таблицами, отчетами и экранными формами). Этот вид блокировок обычно применяется при изменении структуры таблицы.

**Блокировка от записи.** Накладывается в случаях, когда можно использовать таблицу, но без изменения ее структуры или содержимого. Такая блокировка применяется, например, при выполнении операции слияния данных из двух таблиц.

**Предохраняющая блокировка от записи.** Предохраняет объект от наложения на него со стороны других операций полной блокировки, либо блокировки от записи. Этот вид блокировки позволяет тому, кто раньше «захватил» объект, успешно завершить модификацию объекта. Предохраняющая блокировка от записи совместима с аналогичной блокировкой (предохраняющей блокировкой от записи), а также с предохраняющей полной блокировкой. Примером необходимости использования этой блокировки является режим совместного редактирования таблицы несколькими пользователями.

**Предохраняющая полная блокировка.** Предохраняет объект от наложения на него со стороны других операций только полной блокировки. Обеспечивает максимальный уровень совместного использования объектов. Такая блокировка может использоваться, например, для обеспечения одновременного просмотра несколькими пользователями одной таблицы. В группе пользователей, работающих с одной таблицей, эта блокировка не позволит никому изменить структуру общей таблицы.

Что касается БД созданных в среде MS Access, после открытия файла сразу создается файл блокировки (название такое же, как и у исходного, но расширение ldb). По умолчанию, на БД накладывается предохраняющая полная блокировка, т.е. каждый из пользователей может просматривать и изменять данные в таблицах, но не может изменять структуру существующих таблиц, форм, отчетов пока БД пользуется два и более человек. Ограничения на изменения структуры БД снимаются, как только пользователю становится

доступна полная блокировка (монопольный доступ), т.е. как только остальные пользователи заканчивают работу с БД.

Рассмотрим ситуацию, потенциально приводящую к конфликту:

1. Пользователь А и Б открыл базу;
2. Пользователь А начал редактирование 2го поля в 3ем кортеже отношения R1;
3. Пользователь Б начал редактирование 2го поля в 3ем кортеже отношения R1;
4. Пользователь А закончил редактирование;
5. Пользователь Б закончил редактирование.

В результате, изменения вносимые пользователем А просуществовали до завершения редактирования пользователем Б. Более того, пользователь Б затер изменения пользователя А даже не подозревая об этом. Потенциально такая ситуация может приводить к нарушению логики функционирования приложения, использующего БД. Поэтому в MS Access при возникновении рассмотренной ситуации, когда пользователь Б пытается закончить редактирование (п. 5), ему выдается предупреждение о том, что запись уже была изменена, а также предлагается выбрать один из вариантов: закончить редактирование, скопировать внесенные им изменения в буфер и просмотреть измененные (пользователем А) значения полей, отменить редактирование.

В этом случае ответственность за правильное внесение данных перекладывается на пользователей БД и прикладных программных средств, использующих БД.

Кроме того, имеется возможность указать степень детализации блокировки, которую использует Microsoft Access в общей базе данных. Если применяется блокировка на уровне страниц, Microsoft Access блокирует страницу с размером 4К (область памяти, в которой находится запись). При этом изменение записи может привести к блокировке других записей на этой странице. Однако блокировка на уровне страниц обычно обеспечивает более высокое быстродействие.

## **Структура языка SQL**

Все операторы языка SQL можно условно разделить на три группы операторов. Оператор языка запросов – SELECT (был рассмотрен во 2ой лабораторной), операторы языка манипуляции данными (Insert, Update, Delete) и операторы языка определения данных (Create, Drop, Alter).

### **Запросы DML (ЯМД)**

К запросам языка манипуляции данными (Data Manipulation Language) относятся запросы на добавление, удаление и модификацию кортежей.

**Добавление кортежа** производится командой:

*INSERT INTO имя\_таблицы [(*<список столбцов>*)] VALUES (*<список значений>*)*

Список столбцов и список значений указываются через запятую, а значения добавляются в соответствующие столбцы. Если необходимо добавить кортеж целиком (т.е. значения есть для всех полей и их порядок совпадает с порядком полей в отношении), то описание списка столбцов можно опустить.

### **Пример 3.1:**

Три следующих запроса будут верно исполнены для отношения R1 из лабораторной работы №2:

**INSERT INTO R1(ФИО, Дисциплина, Оценка) VALUES («Попова», «БД», 3);**

**INSERT INTO R1 VALUES («Попова», «Моделирование», 3);**

**INSERT INTO R1(ФИО, Дисциплина) VALUES («Бурковский», «Сети ЭВМ»);**

Оператор **удаления данных DELETE** позволяет удалить одну или несколько строк из таблицы в соответствии с условиями, которые задаются для удаляемых строк. Синтаксис оператора DELETE следующий:

*DELETE FROM <имя\_таблицы> [WHERE <условия\_отбора>]*

Если условия отбора не задаются, то из таблицы удаляются все строки. Операция **обновления данных UPDATE** требуется тогда, когда происходят изменения данных, которые надо отразить в базе данных.

Запрос на обновление может изменить сразу целую группу записей. Этот запрос состоит из трех частей:

- Предложение **UPDATE**, которое указывает на обновляемую таблицу;
- Предложение **SET**, задающее данные для обновления;
- Необязательный критерий **WHERE**, ограничивающий число записей, на которые воздействует запрос на обновление.

### **Пример 3.2:**

Изменить на 3 оценку по дисциплине «БД» у студента Миронова в таблице R1 (из лабораторной №2):

**UPDATE R1 SET R1.Оценка = 3**

**WHERE R1.ФИО = «Миронов» AND R1.Дисциплина = «БД»;**

## **Запросы SDL (ЯОД)**

Команды языка определения схемы данных (Schema Definition Language – SDL) представляют собой инструкции SQL, которые позволяют создавать и модифицировать элементы структуры базы данных. Например, используя SDL, можно создавать, удалять таблицы и изменять их структуру, создавать и удалять индексы.

**Создание таблицы.** Оператор создания таблицы имеет следующий вид:

```
CREATE TABLE <имя таблицы>  
(<имя столбца> <тип данных> [NOT NULL]  
[,<имя столбца> <тип данных> [NOT NULL]] ...)
```

Обязательными операндами оператора являются имя создаваемой таблицы и имя хотя бы одного столбца (поля) с указанием типа данных, хранимых в этом столбце.

При создании таблицы для отдельных полей могут указываться некоторые дополнительные правила контроля вводимых в них значений. Например, конструкция NOT NULL (не пустое) служит для определения обязательного поля.

В табл. 3.1 перечислены типы данных, которые можно использовать при создании таблиц, используя Microsoft Jet SDL и предложение CREATE (СУБД Access).

Таблица 3.1. Типы данных полей, доступных в Access

Тип данных	SQL тип
Счетчик	COUNTER
Текстовый	TEXT
Мемо	LONGTEXT
Денежный	CURRENCY
Дата/время	DATETIME
Числовой (одинарное с плавающей точкой)	SINGLE
Числовой (двойное с плавающей точкой)	DOUBLE
Числовой (целое)	INTEGER
Числовой (длинное целое)	LONG
Числовой (байт)	BYTE

С помощью конструкции **CONSTRAINT** можно задать первичный ключ таблицы.

**Пример 3.3:**

Создание таблицы TABL1:

**CREATE TABLE TABL1**

**( [FIL1] COUNTER, [FIL2] TEXT (10),  
[FIL3] CURRENCY, [FIL4] DATETIME,  
[FIL5] BYTE, [FIL6] INTEGER,  
[FIL7] SINGLE, [FIL8] LONG,  
[FIL9] DOUBLE,**

**CONSTRAINT PrimaryKey PRIMARY KEY ([FIL1]) );**

В примере 3.3 поле FIL1 объявлено ключевым, для данного поля создан индекс с именем PrimaryKey.

Похожим образом задается внешний ключ:

**Пример 3.4:**

Создание таблицы TABL2:

```
CREATE TABLE TABL2  
([FIL1] INTEGER, [FIL2] TEXT (10) NOT NULL, [FIL3] CURRENCY,  
[FIL4] LONGTEXT,  
CONSTRAINT PrimaryKey PRIMARY KEY ([FIL1],[FIL2]),  
CONSTRAINT ForeignKey FOREIGN KEY ([FIL1])  
REFERENCES TABL1 ([FIL1]));
```

В данной таблице (пример 3.4) поле FIL1 объявлено внешним ключом. Между таблицами TABL1 и TABL2 устанавливается связь «один-ко-многим» по полю FIL1.

Для **удаления таблиц** служит инструкция

```
DROP TABLE <имя таблицы>
```

Для **модификация структуры таблицы** (добавление, удаление полей, изменения типов полей) используется оператор ALTER TABLE изменения структуры таблицы имеет следующий вид:

```
ALTER TABLE <имя таблицы> MODIFY | ADD | DROP <имя поля>  
[<тип данных>]
```

**Создание индексов.** Помимо создания индексов в процессе формирования таблицы (с помощью предложения CONSTRAINT), можно также создавать индексы уже после того, как таблица сформирована:

```
CREATE [UNIQUE] INDEX <имя индекса> ON <имя таблицы>  
(<имя столбца> [ASC | DESC]  
[, <имя столбца> [ASC | DESC]...)
```

Этот оператор позволяет создать индекс для одного или нескольких столбцов заданной таблицы с целью ускорения выполнения запросных и поисковых операций с таблицей. Для одной таблицы можно создать несколько индексов.

Для **удаления индексов** служит инструкция

```
DROP INDEX <имя индекса> ON <имя таблицы>
```

## **Основы работы с MS Access**

### **Создание макросов в среде MS Access**

Макросом называют набор из одной или более макрокоманд (замкнутая инструкция), выполняющих определенные операции, такие как открытие форм или печать отчетов. Макросы могут быть полезны для автоматизации часто выполняемых задач.

Для того, чтобы создать макрос необходимо открыть объекты «Макросы» и нажать кнопку «Создать» (рис. 3.1).

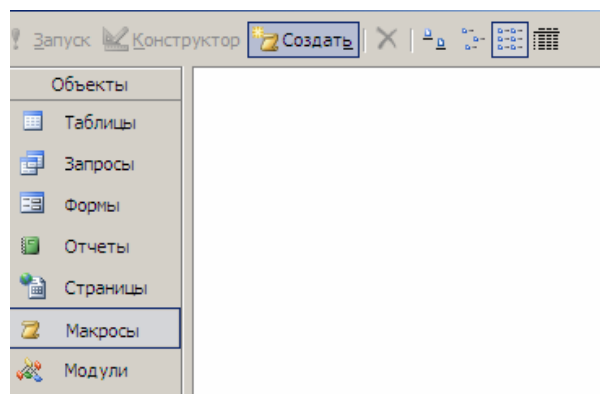


Рисунок 3.1 – Создание макроса в MS Access

На экране появится окно – конструктор макроса, в котором последовательно (по строкам) можно выбрать какие макрокоманды необходимо выполнить (настроить команды можно внизу окна), как это показано на рисунке 3.2.

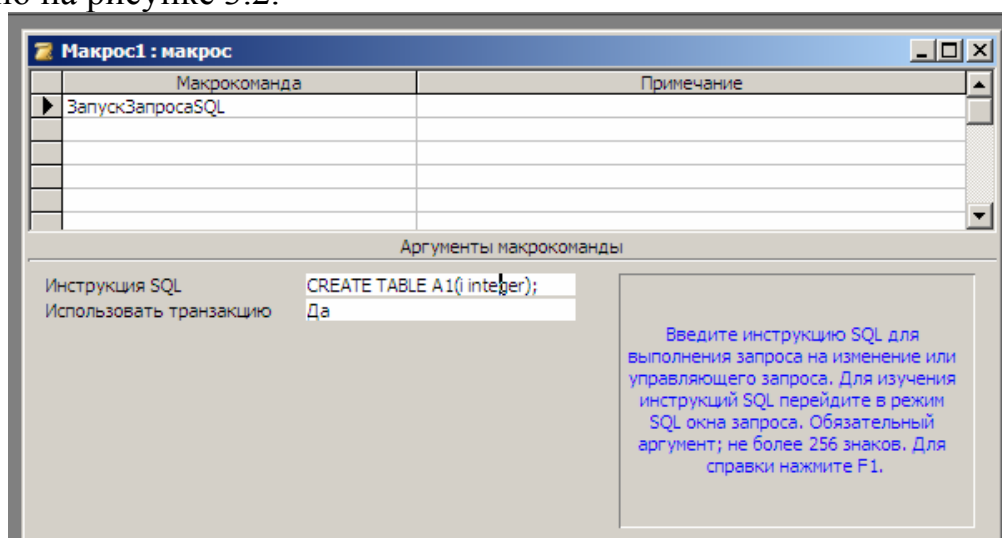


Рисунок 3.2 – Описание макрокоманд макроса в MS Access

После описания всех макрокоманд макрос следует сохранить. Теперь в любой момент для выполнения макроса, необходимо обратиться по его имени.

### Создание форм и отчетов в среде MS Access

СУБД MS Access предоставляет программисту инструментарий для создания форм и отчетов (для пользователя). Соответствующие объекты можно найти среди объектов БД. Доступно как создание форм (отчетов) по определенным таблицам (запросам), так и самостоятельное создание в режиме дизайнера.

### Встроенный язык Visual Basic for Application

СУБД MS Access предоставляет возможность описания процедур на языке высокого уровня Visual Basic for Application (VBA). Этот язык встроен во все программные средства, относящиеся к MS Office, и он позволяет работать с объектами БД через выполнение SQL-запросов. Язык VBA является родственником VB и Basic. Также этот язык является процедурным,

поддерживает деление на модули, поддерживает дизайнер форм. Обеспечивает обработку исключительных ситуаций и выполнение транзакций. Процедуры и функции, описанные в области видимости public, могут также быть использованы при построении SQL-запросов, подобно встроенным функциям СУБД.

Создать модули можно на соответствующей странице объектов MS Access.

Ниже на примере 3.5 представлена процедура, которая используется для выполнения SQL-запросов к БД. Более подробно о синтаксисе языка можно узнать из автономной справки MS Access-a.

### Пример 3.5:

Функция добавления студента в группу для отношения R2 (л/р №2).  
Если студент уже есть – группа обновляется, иначе – создается новая запись:

### Public Function foo3 5(gr student As String, fio student As String)

## ' объявляем переменные

## Dim strSQL As String

## Dim rstSQL As ADODB.Recordset

**Set rstSQL = New ADODB.Recordset**

### Ищем записи о студенте

```
strSQL = "select * from R2 where fio like '" & fio & "' and fio student & '" & fio & "'"
```

**' выполнить запрос**

```
rstSQL.Open strSQL, CurrentProject.Connection
```

**If rstSQL.EOF Then**

**'если ни одной строки не найдено, то добавить новую**

```
strSQL = "insert into R2 (gr,FIO) values('"' & gr_student &
"'",'"' & fio_student & "'")"
```

**Else**

**' если найдена, то обновляем номер группы**

```
strSQL = "update R2 set gr=""" & gr_student & """" where fio like  
"""" & fio_student & """" and gr not like """" & gr_student & """""
```

**End If**

## rstSQL.Close

## DoCmd.SetWarnings False

**DoCmd.RunSQL strSQL** ' выполнить запрос

## DoCmd.SetWarnings True

**End Function;**

“” (дважды двойные кавычки) – используется для вставки в строку “, т.е. вторые двойные кавычки используются как символ экранирования (например, как в С \\\). Комментарии начинаются с символа ‘ – одинарной кавычки. В функции формируется запрос, выполняется и проверяется результат выполнения – если нет полученных кортежей, значит необходимо добавить новую запись «Группа», «Студент», иначе обновить записи для этого студента о его принадлежности к группе.



## Задание на лабораторную работу №3

### Обязательная часть

1. Создать базу данных по любой предметной области (желательно по курсовой работе), которая должна минимум содержать таблицу, состоящую минимум из 6 полей, где обязательно должно присутствовать поле типа date. Для создания таблиц БД **использовать скриптовый файл или макрокоманду**, содержащую набор SQL-команд из части языка SDL;
2. Реализовать процедуры Добавления, Удаления, Поиска и Изменения, с помощью SQL;
3. Организовать оконный интерфейс для функций, созданных на предыдущем этапе (добавления, удаления, поиска и изменения);
4. Поиск должен осуществляться с использованием индексов, т.е. поля, по которым осуществляется поиск, должны быть проиндексированы. Для создания индексов использовать CREATE INDEX.

Выполнение обязательных пунктов = **70%**

**Бонус (+ 15%):** Для получения дополнительных баллов реализовать кодовые поля в основной таблице и справочник(и) для расшифровки этих полей (подобно базе allauto.mdb).

**Бонус (+ 15%):** Организовать механизм авторизации – вход в БД по паролю для нескольких пользователей (статья справки «Пароли (MDB)»).

### Прием работы

Прием происходит при наличии оформленного отчета и работающей БД, созданной в среде MS Access.

### Вопросы

1. На какие части можно разделить язык SQL, какие команды им соответствуют?
2. Что такое механизм блокировки, какой бывает механизм блокировки?
3. Какие существуют варианты для создания индекса у поля в таблице?
4. Для чего используются индексы?
5. Как обновить несколько полей для нескольких кортежей таблицы одним запросом?
6. Что определяет ключевое слово Constraint?
7. Что такое VBA?
8. Можно ли выполнить добавление данных без указания названия полей, в которые добавляются значения? (почему нельзя или как можно)
9. Для чего необходима блокировка, какие конфликтные ситуации могут возникать при отсутствии блокировок, какие пути выхода из этих ситуаций?
10. Чем отличается блокировка на уровне записей от блокировки на уровне страниц?

## **Лабораторная работа №4**

**Тема:** основные функции СУБД, журнализация изменений в базе данных.

**Цель:** моделирования механизмов протоколирования и отката команд.

**Навыки и умения:** реализация алгоритма журнализации, написание клиентского интерфейса, программирование на VBA.

### **Теоретический базис**

#### **Основные функции СУБД**

К основным функциям СУБД можно отнести следующие:

##### **1. Поддержка языков БД**

Как вы уже знаете из предыдущих лабораторных работ, для взаимодействия с БД используются специальные языки: язык определения данных, язык манипулирования данными и язык запросов. В современных СУБД все эти три языка объединены в единый интегрированный язык – SQL. Важнейшая функция СУБД – поддержание этих трех языков (т.е. SQL).

##### **2. Управление транзакциями**

Транзакция – это последовательность операций над БД, рассматриваемых СУБД как единое целое. Либо транзакция успешно выполняется, и СУБД фиксирует (COMMIT) изменения БД, произведенные этой транзакцией, во внешней памяти, либо ни одно из этих изменений никак не отражается на состоянии БД (ROLLBACK). Более подробно механизм транзакций будет рассмотрен в следующих лабораторных работах.

##### **3. Управление данными во внешней памяти.**

##### **4. Управление буферами оперативной памяти.**

##### **5. Обеспечение целостности и безопасности базы данных.**

Целостность базы данных – свойство базы данных, при наличии, которого база данных содержит полную и непротиворечивую информацию, необходимую и достаточную для корректного функционирования приложений. Поддержание целостности базы данных включает проверку целостности и ее восстановление в случае обнаружения противоречий в базе данных. Целостное состояние базы данных описывается с помощью ограничений целостности в виде условий, которым должны удовлетворять хранимые в базе данные. Примером таких условий может служить ограничение диапазонов возможных значений атрибутов объектов, сведения о которых хранятся в базе данных, или отсутствие повторяющихся записей в таблицах реляционных баз данных.

##### **6. Журнализация**

Одним из основных требований к СУБД является надежность хранения данных во внешней памяти. Под надежностью хранения понимается то, что СУБД должна быть в состоянии восстановить последнее согласованное состояние БД после любого аппаратного или программного сбоя. Обычно

рассматриваются два возможных вида аппаратных сбоев: так называемые мягкие сбои, которые можно трактовать как внезапную остановку работы компьютера (например, аварийное выключение питания), и жесткие сбои, характеризующиеся потерей информации на носителях внешней памяти. Примерами программных сбоев могут быть: аварийное завершение работы СУБД (по причине ошибки в программе или в результате некоторого аппаратного сбоя) или аварийное завершение пользовательской программы, в результате чего некоторая транзакция остается незавершенной. Первую ситуацию можно рассматривать как особый вид мягкого аппаратного сбоя; при возникновении последней требуется ликвидировать последствия только одной транзакции.

Понятно, что в любом случае для восстановления БД нужно располагать некоторой дополнительной информацией. Другими словами, поддержание надежности хранения данных в БД требует избыточности хранения данных. Наиболее распространенным методом поддержания такой избыточной информации является ведение журнала изменений БД.

Журнал – это особая часть БД, недоступная пользователям СУБД, в которую поступают записи обо всех изменениях основной части БД. В разных СУБД изменения БД журналируются на разных уровнях: иногда запись в журнале соответствует некоторой логической операции изменения БД (например, операции удаления строки из таблицы реляционной БД), иногда – минимальной внутренней операции модификации страницы внешней памяти; в некоторых системах одновременно используются оба подхода.

Во всех случаях придерживаются стратегии "упреждающей" записи в журнал (так называемого протокола Write Ahead Log - WAL). Грубо говоря, эта стратегия заключается в том, что запись об изменении любого объекта БД должна попасть во внешнюю память журнала раньше, чем измененный объект попадет во внешнюю память основной части БД. Известно, что если в СУБД корректно соблюдается протокол WAL, то с помощью журнала можно решить все проблемы восстановления БД после любого сбоя.

Самая простая ситуация восстановления – индивидуальный откат транзакции. Строго говоря, для этого не требуется общесистемный журнал изменений БД. Достаточно для каждой транзакции поддерживать локальный журнал операций модификации БД, выполненных в этой транзакции, и производить откат транзакции путем выполнения обратных операций, следуя от конца локального журнала. В некоторых СУБД так и делают, но в большинстве систем локальные журналы не поддерживают, а индивидуальный откат транзакции выполняют по общесистемному журналу, для чего все записи от одной транзакции связывают обратным списком (от конца к началу).

При мягком сбое во внешней памяти основной части БД могут находиться объекты, модифицированные транзакциями, не закончившимися к моменту сбоя, и могут отсутствовать объекты, модифицированные транзакциями, которые к моменту сбоя успешно завершились (по причине

использования буферов оперативной памяти, содержимое которых при мягком сбое пропадает). При соблюдении протокола WAL во внешней памяти журнала должны гарантированно находиться записи, относящиеся к операциям модификации обоих видов объектов. Целью процесса восстановления после мягкого сбоя является состояние внешней памяти основной части БД, которое возникло бы при фиксации во внешней памяти изменений всех завершившихся транзакций и которое не содержало бы никаких следов незаконченных транзакций. Для того чтобы этого добиться, сначала производят откат незавершенных транзакций (undo), а потом повторно воспроизводят (redo) те операции завершенных транзакций, результаты которых не отображены во внешней памяти.

Для восстановления БД после жесткого сбоя используют журнал и архивную копию БД. Грубо говоря, архивная копия - это полная копия БД к моменту начала заполнения журнала (имеется много вариантов более гибкой трактовки смысла архивной копии). Конечно, для нормального восстановления БД после жесткого сбоя необходимо, чтобы журнал не пропал.

## Основы работы с MS Access

### Открытие формы при запуске БД в среде MS Access

Взаимодействия пользователя с БД, как правило, не подразумевает возможности исправления структуры БД, а также выполнение пользователем SQL-запросов напрямую. Обычно внутренняя структура БД скрыта от пользователя, для того, чтобы не было возможности случайно испортить данные в БД. Подобное можно сделать и в MS Access, для чего можно использовать функцию открытия формы при запуске.

Для этого, необходимо выполнить пункты меню: «Сервис» -> «Параметры запуска», после чего будет открыта форма, для настройки параметров запуска БД, как показано на рис. 4.1

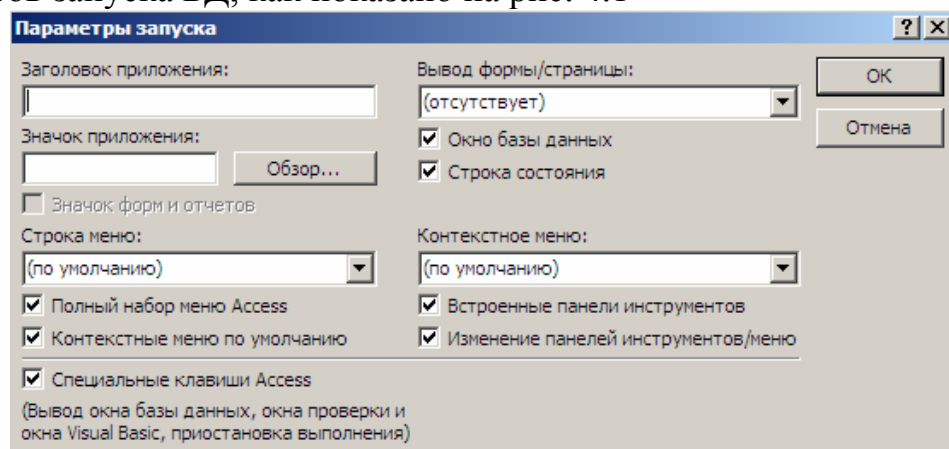


Рисунок 4.1 – Параметры запуска приложения БД в среде MS Access

Теперь, если вы захотите открыть БД в обычном режиме (без параметров запуска), необходимо использовать клавишу SHIFT.

## **Запуск макросов с использованием языка VBA**

Макросы могут быть запущены из процедуры, написанной на языке VBA, так, как показано в примере 4.1

### **Пример 4.1:**

Для запуска макроса из VBA, необходимо выполнить команду RunMacro, параметром которой указать имя макроса:

```
Public Function foo4_1()  
    DoCmd.RunMacro "Macro1"  
End Function;
```

## Задание на лабораторную работу №4

### Обязательная часть

1. Необходимо реализовать главную форму, запускаемую автоматически при открытии БД (для allauto.mdb). Эта форма должна позволять редактировать данные о а/м в таблице AUTO: добавление, удаление, изменение автомобиля (или автомобилей). При запуске приложения в нормальном режиме не выводить окно базы данных.

2. Реализовать протоколирование – журнал изменений. Должны быть реализованы функции отката изменений БД (таких как добавление, удаление, изменение записи). Для этого организовать специальную форму, позволяющую осуществлять:

- "Откат назад" – откат назад на одно изменение в базе (не активен, если не было изменений или выполнены все откаты назад);
- "Откат вперед" – откат вперед на одно изменение (может быть не активен). Не забудьте проиндексировать таблицу MENU.

Выполнение обязательных пунктов = **55%**

**Бонус (+ 15%):** предусмотреть, что при аварийном завершении программы, существует возможность восстановления всей цепочки отката, и лишь только при "нормальном" завершении работы с программой цепочка отката обнуляется (для этого используйте макросы).

**Бонус (+ 30%):** Реализовать процедура поиска для главной формы.

"Поиск" - содержит следующие подпункты (функции):

1. "Параметры" - выдается форма, содержащие перечень не менее 3х кодовых полей (любые, из таблицы AUTO). При выборе любого из этих пунктов выдается перечень возможных значений, который может принять данное кодовое поле (выпадающий список значений из menu). У значений могут быть свои подзначения, поэтому необходимо организовать рекурсивную (или циклическую) процедуру обхода дерева значений (словаря данных) – т.е. для каждого поля выводить не только его значения, но и подзначения. У ВСЕХ ДОЛЖНО БЫТЬ КОДОВОЕ ПОЛЕ "МАРКА ТРАНСПОРТНОГО СРЕДСТВА". При выборе значений, заносить их в строку поиска, типа: <код. поле>=<значение> AND
2. "Строка поиска" - выводит на экран содержимое строки поиска.
3. "Очистить строку поиска" - обнуляет строку поиска.
4. "Найти" – поиск записей в таблице auto в соответствии со строкой поиска, если строка поиска пуста, то выводятся все записи.

Все поля, которые используется для поиска, должны быть расшифрованы.

## **Прием работы**

Прием происходит при наличии оформленного отчета и работающей БД, созданной в среде MS Access.

## **Вопросы**

1. Назовите основные функции СУБД.
2. Какие бывают языки взаимодействия с БД?
3. Что такое транзакция?
4. Что такое журнализация?
5. Что понимается под понятием «целостность БД»?
6. Какие бывают виды сбоев? Охарактеризуйте их.
7. Что такое протокол WAL?

## Лабораторная работа №5

**Тема:** поддержка темпоральности изменяемых данных.

**Цель:** написание программного интерфейса для хранения изменений данных в БД и организации доступа к ним.

**Навыки и умения:** протоколирование и работа с изменениями кортежей в БД.

### Задание на лабораторную работу №5

#### Обязательная часть

Реализовать возможность просматривать содержимое базы auto на заданную дату для одного поля. Для этого доработать программу протоколирования (л.р.№4), добавив пункты:

1. "Переход на заданную дату" – выводятся только те данные в базе, которые были актуальны на заданную дату(выводится запрос на ввод даты);

2. "Переход на текущую дату" – аналогично п.1, только на текущую дату время.

В программе должны быть объявлены 2 переменных: текущая дата и заданная дата. При запуске программы спрашивается текущая дата (по умолчанию действительная текущая дата). На экране должна отображаться текущая дата и заданная дата (по умолчанию равна текущей дате).

Процедура поиска ищет данные удовлетворяющее заданному условию и актуальные на дату, указанную в переменной, содержащей заданную дату.

При откате на заданную дату доступен лишь просмотр базы и поиск.

Выполнение обязательного задания = **55%**

**Бонус (+ 45%):** Реализовать возможность просматривать содержимое базы AUTO на заданную дату для всех полей базы.

#### Прием работы

Прием происходит при наличии оформленного отчета и работающей БД, созданной в среде MS Access.

Вопросы к 5-ой (заключительная л/р для практикума с средой MS Access) составлены из вопросов к предыдущим лабораторным работам.

#### Вопросы

1. Что такое база данных?
2. Что такое система баз данных?
3. Что такое система управления базами данных?
4. Основное назначение?
5. Основные компоненты СУБД?
6. Что подразумевает понятие абстрагирование в СУБД?
7. Какие существуют уровни абстракции в структурных данных?
8. Опишите уровень представления



9. Опишите концептуальный уровень
10. Опишите физический уровень
11. Виды связей
12. Что такое отношение (таблица) в реляционной модели СУБД?
13. Что такое домен в реляционной модели СУБД?
14. Что такое атрибут (поле) в реляционной модели СУБД?
15. Что такое картеж (хранимая запись) в реляционной модели СУБД?
16. Что такое первичный ключ?
17. Что такое потенциальный ключ?
18. Что такое внешний ключ?
19. Что такое SQL, назначение языка SQL?
20. Назначение команды SELECT?
21. Что такое внешнее и внутреннее объединение, чем отличаются?
22. Что такое левое, правое и полное объединение?
23. Что такое перекрестный запрос?
24. Как применить агрегатную функцию?
25. Для чего в стандарт SQL2 были введены объединения?
26. Чем отличается использование WHERE от HAVING?
27. Чем отличается использование DISTINCT от группировки?
28. На какие части можно разделить язык SQL, какие команды им соответствуют?
29. Что такое механизм блокировки, какой бывает механизм блокировки?
30. Какие существуют варианты для создания индекса у поля в таблице?
31. Для чего используются индексы?
32. Как обновить несколько полей для нескольких кортежей таблицы одним запросом?
33. Что определяет ключевое слово Constraint?
34. Что такое VBA?
35. Можно ли выполнить добавление данных без указания названия полей, в которые добавляются значения? (почему нельзя или как можно)
36. Для чего необходима блокировка, какие конфликтные ситуации могут возникать при отсутствии блокировок, какие пути выхода из этих ситуаций?
37. Чем отличается блокировка на уровне записей от блокировки на уровне страниц?
38. Назовите основные функции СУБД.
39. Какие бывают языки взаимодействия с БД?
40. Что такое транзакция?
41. Что такое журнализация?
42. Что понимается под понятием «целостность БД»?
43. Какие бывают виды сбоев? Охарактеризуйте их.
44. Что такое протокол WAL?

## **Лабораторная работа №6**

**Тема:** СУБД PostgreSQL, нетривиальные возможности.

**Цель:** познакомиться с интерфейсом взаимодействия с PostgreSQL, а также научиться применять некоторые нетривиальные возможности СУБД.

**Навыки и умения:** работа с `psql`, создание объектно-реляционных связей, использование ограничений в таблицах, использование массивов, использование последовательностей, `backup` и `restore` БД.

### **Теоретический базис**

#### **Знакомство с PostgreSQL**

PostgreSQL — объектно-реляционная система управления базами данных (ОРСУБД), разработка которой в различных формах ведется с 1977 года. Работа началась с проекта Ingres в Калифорнийском университете (Беркли). Затем проект Ingres был переведен на коммерческую разработку в корпорации Relational Technologies/Ingres.

В 1986 году другая группа, которую возглавлял Майкл Стоунбрейкер (Michael Stonebraker) из Беркли, продолжила работу над Ingres и создала объектно-реляционную СУБД Postgres. В 1996 году из-за усовершенствования пакета и перехода на распространение с открытыми исходными текстами было принято новое название — PostgreSQL (в течение непродолжительного времени использовалось название Postgres95). В настоящее время над проектом PostgreSQL активно работает группа разработчиков со всего мира.

PostgreSQL считается самой совершенной СУБД, распространяемой на условиях открытых исходных текстов. В PostgreSQL реализованы многие возможности, традиционно встречавшиеся только в масштабных коммерческих продуктах. Проект PostgreSQL распространяется на условиях открытых исходных текстов.

По данным многих тестов проводимых для СУБД PostgreSQL не значительно уступает по своей производительности Oracle (около 15%). При этом одна из самых больших БД в мире – БД Yahoo (в 2008 году объем СУБД составлял два петабайта) работает на модифицированной версии СУБД PostgreSQL. Одно из самых крупных изменений: ориентация на поколонное хранение вместо традиционного построчного, что замедляет запись на диск, но обеспечивает лучшую скорость доступа к данным для аналитических целей.

#### **Объектно-реляционная СУБД**

PostgreSQL относится к категории объектно-реляционных систем управления базами данных (ОРСУБД). Модель ОРСУБД представляет собой усовершенствование более традиционной модели реляционной системы управления базами данных (РСУБД). В РСУБД логически связанные данные

хранятся в двумерных структурах, называемых таблицами. Данные могут состоять из элементов, относящихся к различным стандартным типам — целые и вещественные числа, символы, строки, дата/время. В таблице элементы данных образуют «решетку» из столбцов (полей) и строк (записей). Одной из главных особенностей реляционной модели является ее концептуальная простота, причем это может считаться как ее главным достоинством, так и главным недостатком.

Объектно-реляционная специфика PostgreSQL дополняет традиционную реляционную модель данных многочисленными усовершенствованиями. К их числу относится поддержка массивов (хранения нескольких элементов в одном поле), наследования (связей типа «предок—потомок» между таблицами) и функций (программных методов, вызываемых командами SQL). В PostgreSQL также предусмотрены возможности расширения типов данных и использования процедурных языков.

Вследствие объектно-реляционной ориентации таблицы иногда называются классами, а записи и поля могут соответственно именоваться экземплярами (instances) и атрибутами (attributes).

## SQL в PostgreSQL

Система PostgreSQL, как и большинство сетевых СУБД, основана на парадигме «клиент-сервер». Центральное место в PostgreSQL занимает процесс postmaster, предназначенный не для прямого взаимодействия с пользователем, а для обслуживания подключений со стороны различных клиентов.

Существует несколько интерфейсов, через которые клиент подключается к процессу postmaster. В примерах этой книги используется psql — самый универсальный и доступный клиент, входящий в комплект поставки PostgreSQL. Клиент psql работает в режиме командной строки.

В psql существует два способа ввода и исполнения запросов: в интерактивном режиме запросы обычно вводятся непосредственно в приглашении командной строки; команда psql \i читает файл локальной файловой системы и использует его содержимое в качестве входных данных.

Все SQL команды принято записывать в верхнем регистре, а имена таблиц, полей (и т.д.) в нижнем регистре. Если необходимо записать имя таблицы в верхнем (или смешанном) регистре, то необходимо название заключить в двойные кавычки. Это связано с тем, что перед разбором строки запроса сервер postgresql переводит весь запрос в нижний регистр. Для того, чтобы сообщить серверу, что имя таблицы, схемы, поля (или другое) не надо переводить в нижний регистр — служат двойные кавычки. Существует также возможность создавать объекты БД с именами, которые являются ключевыми словами SQL. Для этого также необходимо заключить название объекта в обрамляющие двойные кавычки, как это показано в примере 6.1

Для интерактивной работы с базой в оконном режиме в пакет поставки PostgreSQL входит утилита PgAdmin.

### Пример 6.1:

Создание отношения группы:

```
CREATE TABLE "group"  
(  
    "name" text,  
    num integer,  
    "Fio" text  
);
```

### Ограничения в таблицах

Ранее нами уже были рассмотрены некоторые ограничения, используемые в среде MS Access. Теперь рассмотрим ограничения, использование которых предоставляет нам PostgreSQL.

Ограничение (constraint) представляет собой особый атрибут таблицы, который устанавливает критерии допустимости для содержимого ее полей. Соблюдение этих правил помогает предотвратить заполнение базы ошибочными или неподходящими данными.

Формат установки ограничений следующий:

```
[ CONSTRAINT ограничение ]  
{ NOT NULL | UNIQUE | PRIMARY KEY | DEFAULT значение | CHECK ( условие ) |  
REFERENCES таблица [ ( поле ) ]  
[ MATCH FULL | MATCH PARTIAL ]  
[ ON DELETE операция ]  
[ ON UPDATE операция ]  
[ DEFERRABLE | NOT DEFERRABLE ]  
[ INITIALLY DEFERRED | INITIALLY IMMEDIATE ] }
```

Определение следует в команде CREATE TABLE сразу же за типом ограничиваемого поля и предшествует запятой, отделяющей его от следующего поля. Ограничения могут устанавливаться для любого количества полей, а ключевое слово CONSTRAINT и идентификатор ограничение не обязательны.

Существует шесть типов ограничений полей, задаваемых при помощи специальных ключевых слов. Некоторые из них косвенно устанавливаются при создании ограничений другого типа. Некоторые из типов ограничений полей перечислены ниже (более подробно [6]):

- NOT NULL. Поле не может содержать псевдозначение NULL. Ограничение NOT NULL эквивалентно ограничению CHECK (поле NOT NULL).

- UNIQUE. Поле не может содержать повторяющиеся значения. Следует учитывать, что ограничение UNIQUE допускает многократное вхождение псевдозначений NULL, поскольку формально NULL не совпадает ни с каким другим значением.

- PRIMARY KEY. Автоматически устанавливает ограничения UNIQUE и NOT NULL, а для заданного поля создается индекс. В таблице может устанавливаться только одно ограничение первичного ключа.

- DEFAULT значение. Пропущенные значения поля заменяются заданной величиной. Значение по умолчанию должно относиться к типу данных, соответствующему типу поля. В PostgreSQL 7.1.x значение по умолчанию не может задаваться при помощи подзапроса.

- CHECK условие. Команда INSERT или UPDATE для записи завершается успешно лишь при выполнении заданного условия (выражения, возвращающего логический результат). При установке ограничения поля в секции CHECK может использоваться только поле, для которого устанавливается ограничение.

- REFERENCES. Это ограничение состоит из нескольких секций, которые перечислены ниже.

- о REFERENCES таблица [ ( поле ) ]. Входные значения ограничиваемого поля сравниваются со значениями другого поля в заданной таблице. Если совпадения отсутствуют, команда INSERT или UPDATE завершается неудачей. Если параметр поле не указан, проверка выполняется по первичному ключу. Ограничение REFERENCES похоже на ограничение таблицы FOREIGN KEY, описанное в следующем пункте этого подраздела. Действительно, между этими ограничениями есть много общего.

- о ON DELETE операция. При выполнении команды DELETE для заданной таблицы с ограничиваемым полем выполняется одна из следующих операций: NO ACTION (если удаление приводит к нарушению целостности ссылок, происходит ошибка; используется по умолчанию, если операция не указана), RESTRICT (аналогично NO ACTION), CASCADE (удаление всех записей, содержащих ссылки на удаляемую запись), SET NULL (поля, содержащие ссылки на удаляемую запись, заменяются псевдозначениями NULL), SET DEFAULT (полям, содержащим ссылки на удаляемую запись, присваивается значение по умолчанию).

- о ON UPDATE операция. При выполнении команды UPDATE для заданной таблицы выполняется одна из операций, описанных выше. По умолчанию используется значение NO ACTION. Если выбрана операция CASCADE, все записи, содержащие ссылки на обновляемую запись, обновляются новым значением (вместо удаления, как в случае с ON DELETE CASCADE).

В ограничениях таблиц, в отличие от ограничений полей, могут участвовать сразу несколько полей таблицы. Синтаксис ограничения таблицы:

```
[ CONSTRAINT ограничение ]  
{ UNIQUE ( поле [...] ) |  
PRIMARY KEY ( поле [ . ... ] ) |  
CHECK ( условие ) |
```

*FOREIGN KEY ( поле [. ... ] )*  
*REFERENCES таблица [ ( поле [. ] ) ]*  
*[ MATCH FULL | MATCH PARTIAL ]*  
*[ ON DELETE операция ]*  
*[ ON UPDATE операция ]*  
*[ DEFERRABLE | NOT DEFERRABLE ]*  
*[ INITIALLY DEFERRED | INITIALLY IMMEDIATE ]*

Секция CONSTRAINT ограничение определяет необязательное имя. Ограничениям рекомендуется присваивать содержательные имена вместо автоматически сгенерированных имен, не несущих никакой полезной информации. В будущем имя также может пригодиться и для удаления ограничения.

В примере 6.2 первое ограничение, pkey, относится к типу PRIMARY KEY и устанавливается для таблицы по полю isbn. Оно практически не отличается от ограничения PRIMARY KEY для поля, поскольку список в данном примере состоит всего из одного поля.

**Пример 6.2:**

Использование ограниченной таблицы:

```
CREATE TABLE editions  
(isbn text, bookid integer, edition integer, publisherid integer, publication  
date, type char,  
CONSTRAINT pkey PRIMARY KEY (isbn),  
CONSTRAINT integrity CHECK (bookid IS NOT NULL  
AND edition IS NOT NULL).  
CONSTRAINT book_exists FOREIGN KEY (book_id)  
REFERENCES books (id)  
ON DELETE CASCADE  
ON UPDATE CASCADE):
```

Ограничение integrity гарантирует, что поля book\_id и edition не содержат псевдозначения NULL.

Наконец, ограничение book\_exists при помощи конструкций FOREIGN KEY и REFERENCES гарантирует, что значение поля book\_id встречается в поле id таблицы books. Более того, поскольку в секциях ON DELETE и ON ACTION встречается ключевое слово CASCADE, любые модификации поля id в таблице books приведут к каскадным изменениям записей в таблице editions, а при удалении записей из таблицы books будут удалены соответствующие записи таблицы editions.

Для этих ограничений в базе данных автоматически строится индекс editions\_pkey по полю isbn, а также создается триггер. Индекс обеспечивает выполнение ограничения PRIMARY KEY, а триггер относится к ограничению FOREIGN KEY.

## Создание объектно-реляционных связей

В PostgreSQL поддерживается механизм создания объектно-реляционных связей, называемый наследованием. Таблица может наследовать некоторые атрибуты своих полей от одной или нескольких других таблиц, что приводит к созданию отношений типа «предок—потомок». В результате производные таблицы («потомки») обладают теми же полями и ограничениями, что и их базовые таблицы («предки»), а также дополняются собственными полями.

При составлении запроса к базовой таблице можно потребовать, чтобы запрос произвел выборку только из самой таблицы или же просмотрел как таблицу, так и ее производные таблицы. С другой стороны, в результаты запроса к производной таблице никогда не включаются записи из базовой таблицы.

Производная таблица создается командой SQL CREATE TABLE, в которую включается секция INHERITS. Секция состоит из ключевого слова INHERITS и имени базовой таблицы (или нескольких таблиц).

*CREATE TABLE производная\_таблица определение  
INHERITS ( базовая\_таблица [, ...] )*

В этом определении производная\_таблица — имя создаваемой таблицы, определение — полное определение таблицы со всеми стандартными секциями команды CREATE TABLE, а базовая\_таблица — таблица, структура которой наследуется новой таблицей. Дополнительные имена базовых таблиц перечисляются через запятую.

### **Пример 6.3:**

Создание наследования для таблицы В от таблицы А:

```
CREATE TABLE b (b_text text)  
INHERITS (a);
```

Связь общих полей базовой и производной таблиц не ограничивается чисто косметическими удобствами. Данные, занесенные в производную таблицу, присутствуют и в родительской таблице. Впрочем, в родительской таблице видны только три унаследованных поля. В запрос к базовой таблице можно включить ключевое слово ONLY, которое указывает, что данные производных таблиц исключаются из результатов запроса.

**Следует хорошо понимать, что данные в действительности не заносятся в базовую таблицу, а лишь становятся видимыми в ней благодаря отношению наследования.**

Наследование может приводить к видимому нарушению ограничений. Например, значение поля, для которого установлено ограничение уникальности, может повторяться в данных производных таблиц. Применение наследования требует осторожности, поскольку производная таблица формально не нарушает ограничений, хотя при выборке из базовой таблицы без ключевого слова ONLY может показаться обратное.

## Массивы

Поля данных PostgreSQL вместо отдельных величин могут содержать конструкции, называемые массивами. Массив сам по себе не является самостоятельным типом данных, а лишь расширяет любой другой тип данных PostgreSQL.

Чтобы создать простейшее поле-массив, включите в команду CREATE TABLE пару квадратных скобок после имени поля. Квадратные скобки показывают, что вместо одного значения в поле может храниться массив указанного типа. Дополнительные квадратные скобки определяют многомерные массивы, то есть «массивы массивов».

### Пример 6.4:

Создание таблицы с полем-массивом:

```
CREATE TABLE favorite_books (employee_id integer,  
                                books text[]);
```

В PostgreSQL предусмотрен специальный синтаксис вставки нескольких значений в одно поле. Этот синтаксис основан на определении массивов-констант.

Обобщенная форма массива-константы выглядит так:

```
'{ "текст" [, ...] }'      -- массив строк
```

```
'{ число [, ...] }'      -- числовой массив
```

В этих примерах использованы строковые и числовые массивы, но поле может определяться как массив произвольного типа (включая типы boolean, date и time). Как правило, если для описания величины в скалярном контексте должны использоваться апострофы (например, в строковых константах или данных типа timestamp), в контексте массива эта величина заключается в кавычки. Даже при вставке одного элемента массив заключается в фигурные скобки.

### Пример 6.5:

Вставка с использованием массивов-констант:

```
INSERT INTO favorite_books VALUES  
    (102, '{"The Hitchhiker\'s Guide to the Galaxy11"}');  
INSERT INTO favorite_books VALUES  
    (103, '{"The Hobbit", "Kitten. Squared"}');
```

При выборке из поля-массива весь массив возвращается в формате константы.

Популярность массивов в значительной степени обусловлена тем фактом, что к отдельным элементам можно обращаться при помощи индексов — целых чисел, заключенных в скобки и описывающих позицию искомого элемента в массиве. В отличие от таких языков программирования, как C, в PostgreSQL индексация в массивах начинается с 1, а не с 0.

При указании индекса несуществующего элемента массива выборка возвращает NULL. Обычно для обработки таких ситуаций используется конструкция IS NOT NULL.



### **Пример 6.6:**

Выборка из поля-массива по индексу (для таблицы, полученной из предыдущих примеров):

```
SELECT books[2] FROM favorite_books;
```

Вернет 2 строки, одна из которых будет пустой. Для того, чтобы избежать этого, необходимо добавить условие в секцию WHERE.

```
SELECT books[2] FROM favorite_books  
WHERE books[2] IS NOT NULL;
```

В PostgreSQL также поддерживается возможность создания срезов при выборке из массива. Срез аналогичен обычному обращению к элементам по индексу, но он описывает интервал значений. Срез задается парой целочисленных индексов, разделенных двоеточием и заключенных в квадратные скобки.

### **Пример 6.7:**

Запрос среза с первого по второй элемент (включительно):

```
SELECT books[1:2] FROM favorite_books;
```

Чтобы узнать количество значений, хранящихся в массиве, следует воспользоваться функцией `array_dims()`.

## **Автоматизация стандартных процедур**

В лабораторной работе будут рассмотрены две категории расширений, позволяющих автоматизировать часто выполняемых операций с БД: последовательности и триггеры.

### **1) Последовательности**

Последовательностью (sequence) в PostgreSQL называется объект базы данных, который фактически представляет собой автоматически увеличивающееся число. В других СУБД последовательности часто называются счетчиками. Последовательность определяется текущим числовым значением и набором характеристик, определяющих алгоритм автоматического увеличения (или уменьшения) используемых данных.

Наряду с текущим значением в определение последовательности также включается минимальное значение, максимальное значение и приращение. Обычно приращение равно 1, но оно также может быть любым целым числом.

На практике последовательности не рассчитаны на прямой доступ из программы. Работа с ними осуществляется через специальные функции PostgreSQL, предназначенные для увеличения, присваивания или получения текущего значения последовательности.

Последовательности создаются командой SQL `CREATE SEQUENCE` с положительным или отрицательным приращением. В этом определении единственный обязательный параметр *последовательность* определяет имя создаваемой последовательности. Значения последовательности представляются типом `integer`, поэтому максимальное и минимальное

значения должны лежать в интервале от 2 147 483 647 до -2 147 483 647.  
Синтаксис команды CREATE SEQUENCE:

*CREATE SEQUENCE* последовательность  
[ *INCREMENT* приращение ]  
[ *MINVALUE* минимум ]  
[ *MAXVALUE* максимум ]  
[ *START* начало ]  
[ *CACHE* кэш ]  
[ *CYCLE* ]

Ниже описаны некоторые необязательные секции команды CREATE SEQUENCE (полное описание в [6]).

- **INCREMENT** приращение. Числовое изменение текущего значения последовательности. Отрицательное приращение создает убывающую последовательность. По умолчанию приращение равно 1.

- **MINVALUE** минимум. Минимальное допустимое значение последовательности.

- **MAXVALUE** максимум. Максимальное допустимое значение последовательности.

- **START** начало. Начальное значение последовательности, которым является любое целое число в интервале между минимальным и максимальным значениями.

- **CACHE** кэш. Возможность предварительного вычисления и хранения значений последовательности в памяти. Кэширование ускоряет доступ к часто используемым последовательностям. Минимальное значение, заданное по умолчанию, равно 1; увеличение объема кэша приводит к увеличению числа кэшируемых значений.

- **CYCLE**. При достижении нижнего или верхнего порога последовательность продолжает генерировать новые значения. В этом случае она переходит к минимальному значению (для возрастающих последовательностей) или к максимальному значению (для убывающих последовательностей).

К последовательности можно обратиться командой SELECT, как к таблице или представлению (хотя такая возможность используется относительно редко). При составлении запроса к последовательности в списке выборки вместо полей указываются атрибуты последовательности.

Операции с последовательностями:

- **nextval**('последовательность') – увеличивает текущее значение последовательности и возвращает новое;
- **currval** (' последовательность') – возвращает значение, полученное при последнем вызове nextval. Если в текущем сеансе nextval не вызывалась, то currval не сможет вернуть значение;
- **setval** ('последовательность', n) – присваивает текущее значение последовательности, следующий вызов nextval вернет значение n+приращение;

- `setval (' последовательность', n, b)` – также присваивает текущее значение последовательности. Если третий параметр (b) равен `false`, то следующий вызов `nextval` вернет `n`, иначе – `n+преращение`.

#### **Пример 6.8:**

Создание отношение с автоматически увеличивающимся полем (последовательность необходимо создать заранее):

```
CREATE TABLE shipments (id integer DEFAULT  
nextval('shipments_ship_id_seq') PRIMARY KEY,  
customer_id integer, isbn text, ship_date timestamp);
```

Простая установка ограничения `DEFAULT` не гарантирует его применения. Пользователь способен вручную задать любое значение, что может привести к потенциальному нарушению уникальности в будущем. Для предотвращения конфликтов можно воспользоваться триггером.

### **2) Триггеры**

Довольно часто перед некоторыми событиями SQL или после них должны выполняться определенные операции — например, проверка логической целостности данных. В PostgreSQL поддерживаются нестандартные расширения, называемые триггерами (trigger) и упрощающие взаимодействие приложения с базой данных. Триггер определяет функцию, которая должна выполняться до или после некоторой операции с базой данных. Триггеры реализуются на языке C, PL/pgSQL или любом другом функциональном языке (кроме SQL), который может использоваться в PostgreSQL для определения функций.

Триггер создается на основе существующей функции. Триггеры могут вызывать функции, написанные на любом языке, но за одним исключением: функция не может быть полностью реализована на SQL.

Синтаксис определения триггера выглядит так:

```
CREATE TRIGGER триггер { BEFORE | AFTER } { событие [ OR  
событие ] } ON таблица  
FOR EACH { ROW | STATEMENT }  
EXECUTE PROCEDURE функция ( аргументы )
```

Ниже приводятся краткие описания компонентов этого определения.

- `CREATE TRIGGER` триггер. В аргументе триггер указывается произвольное имя создаваемого триггера. Имя может совпадать с именем триггера, уже существующего в базе данных — при условии, что этот триггер установлен для другой таблицы.

- `{ BEFORE | AFTER }`. Ключевое слово `BEFORE` означает, что функция должна выполняться перед попыткой выполнения операции, включая все встроенные проверки ограничений данных, реализуемые при выполнении команд `INSERT` и `DELETE`. Ключевое слово `AFTER` означает, что функция вызывается после завершения операции.

- `{ событие [ OR событие ... ] }`. События SQL, поддерживаемые в PostgreSQL. При перечислении нескольких событий в качестве разделителя используется ключевое слово `OR`.

- ON таблица. Имя таблицы, модификация которой заданным событием приводит к срабатыванию триггера.
- FOR EACH { ROW | STATEMENT }. Ключевое слово, следующее за конструкцией FOR EACH и определяющее количество вызовов функции при наступлении указанного события. Ключевое слово ROW означает, что функция вызывается для каждой модифицируемой записи. Если функция должна вызываться всего один раз для всей команды, используется ключевое слово STATEMENT.
- EXECUTE PROCEDURE функция (аргументы). Имя вызываемой функции аргументами.

Механизм ограничений PostgreSQL позволяет реализовать простое сравнение данных со статическими значениями, но иногда проверка входных данных должна производиться по более сложным критериям. Это типичный пример ситуации, в которой удобно воспользоваться триггером.

#### Пример 6.9:

Создание триггера. Срабатывает при попытке вставить или обновить данные в таблицу shipments и приводит к выполнению функции check\_shipment\_addition():

```
CREATE TRIGGER checkshipment  
BEFORE INSERT OR UPDATE  
ON shipments FOR EACH ROW  
EXECUTE PROCEDURE check_shipment_addition();
```

Удаление триггера происходит с помощью команды Drop Trigger:

*DROP TRIGGER имя ON таблица*

Работа с триггерами предмет для рассмотрения в следующей лабораторной работе.

### Использование PgAdmin для переноса БД

PgAdmin – средство, обеспечивающее удобный оконный интерфейс для работы с базами данных для PostgreSQL. В данной лабораторной работе будут рассмотрены операции создания backup-а (dump-а) базы данных, а также ее восстановления – эти операции необходимо использовать при переносе БД с одного сервера на другой. Для создания backup-а:

1. Запустите PgAdmin и настройте подключение к БД.
2. Затем, выберите БД и вызовите контекстное меню нажатием правой клавиши мыши и выберите «Backup»(рис 6.1).

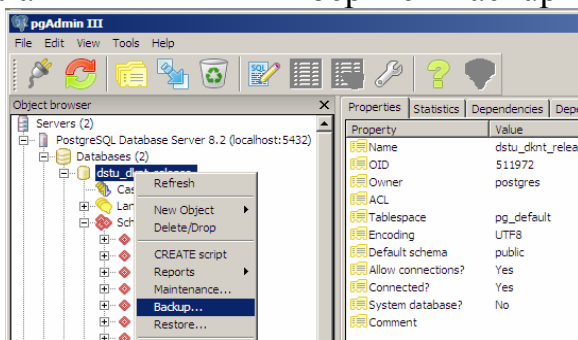


Рисунок 6.1 – Вызов Backup-а из контекстного меню БД в PgAdmin

3. В появившемся окне выберите путь к файлу, в котором будет сохранен backup, а также (если это необходимо) другие настройки (рис 6.2) и нажмите «ОК»

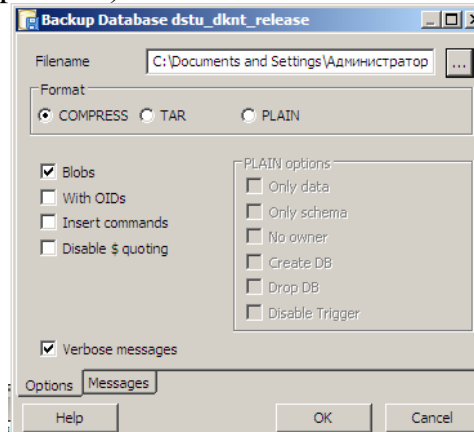


Рисунок 6.2 – Форма для создания backup-а

4. Дождитесь завершения процедуры создания backup-а.

Восстановление БД корректно проходит только в пустую БД, поэтому удалите (или переименуйте) БД, если у вас уже имеется БД с таким именем перед восстановлением, а затем следуйте инструкции:

1. Создайте пустую БД с именем восстанавливаемой БД
2. Для нее вызовите контекстное меню и выберите «Restore»

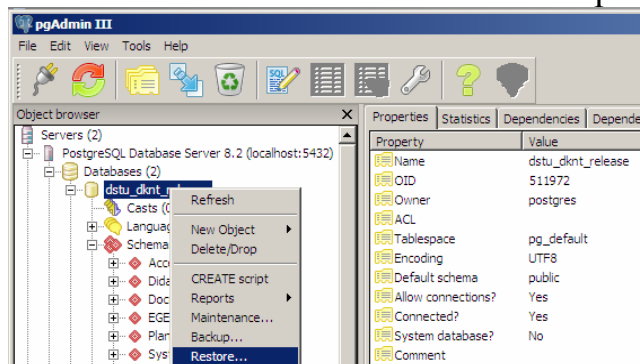


Рисунок 6.3 – Вызов Restore из контекстного меню БД в PgAdmin

3. В появившемся окне выберите путь к файлу с копией БД и, выбрав необходимые настройки, нажмите «Ок» (рис. 6.4)

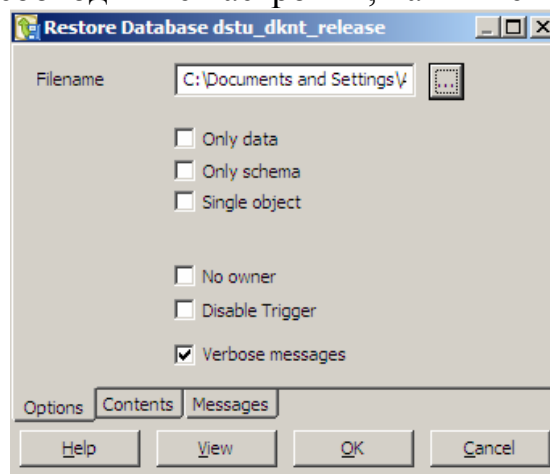


Рисунок 6.4 – Форма для восстановления из backup

4. После окончания нажмите «Done»

## **Задание на лабораторную работу №6**

Выберите предметную область (можно из л/р №1) и опишите структуру БД, используя SQL-запросы. К обязательным требованиям относится использование:

1. объектно-реляционных связей;
2. ограничений в таблицах;
3. массивов;
4. последовательностей;
5. а также backup и restore БД для переноса с домашнего ПК.

Прием работы производится только, если она удовлетворяет всем требованиям.

### **Прием работы**

Прием происходит при наличии оформленного отчета и работающей БД.

### **Вопросы**

1. Что такое PostgreSQL, какой язык использует в качестве языка БД, к какому классу ПО (открытое или закрытое) относится? Какая архитектура? Какие клиентские приложения входят в пакет?
2. Как организуется объектно-реляционные связи в СУБД postgresQL и какие особенности организации могут приводить к визуальному нарушению ограничений установленных в таблицах?
3. Что такое ограничения полей, ограничения таблиц? Как они используются и для чего?
4. Как использовать поля-массивы: как обращаться к элементам массивов, как создавать массивы?
5. Автоматизация стандартных процедур.
6. Что такое последовательности? Как могут быть использованы?
7. Что такое триггеры? На каких языках могут быть реализованы?

## Лабораторная работа №7

**Тема:** Хранимые процедуры на языке PL/pgSQL.

**Цель:** познакомиться с возможностями языка PL/pgSQL.

**Навыки и умения:** написание хранимых процедур на языке PL/pgSQL, создания триггерных функций.

### Теоретический базис

PL/pgSQL относится к семейству процедурных языков и обладает определенным сходством с процедурным языком Oracle, PL/SQL. Процедурным языком называется язык программирования, в котором желаемый результат достигается последовательностью шагов.

Язык PL/pgSQL позволяет группировать на сервере код SQL и программные команды, что приводит к снижению затрат сетевых и коммуникационных ресурсов, обусловленных частыми запросами данных со стороны клиентских приложений и выполнением логической обработки этих данных на удаленных хостах.

В программах PL/pgSQL могут использоваться все типы данных, операторы и функции PostgreSQL. «SQL» в названии PL/pgSQL указывает на то, что программист может напрямую использовать команды языка SQL в своих программах. Использование SQL в коде PL/pgSQL расширяет возможности, а также повышает гибкость и быстродействие программ. Несколько команд SQL в программном блоке PL/pgSQL выполняются за одну операцию вместо обычной обработки каждой команды.

Другой важной особенностью PL/pgSQL является хорошая адаптируемость программ; функции языка совместимы со всеми платформами, на которых работает СУБД PostgreSQL.

### Структура языка

Язык PL/pgSQL имеет относительно простую структуру, что объясняется в основном тем, что каждый логически обособленный фрагмент кода существует в виде функции. Регистр символов в именах функций PL/pgSQL не учитывается. В ключевых словах и идентификаторах допускается использование произвольных комбинаций символов верхнего и нижнего регистров.

### Блоки

Программы PL/pgSQL состоят из блоков. Такой метод организации программного кода обычно называется блочной структурой. Программные блоки вводятся в командах SQL CREATE FUNCTION, которые используются для определения функций PL/pgSQL в базах данных PostgreSQL. Команда CREATE FUNCTION определяет имя функции, типы ее аргументов и

возвращаемого значения. Основной блок функции начинается с секции объявлений. Все переменные объявляются (а также могут инициализироваться значениями по умолчанию) в секции объявления программного блока. В объявлении указывается имя и тип переменной. Секция объявлений обозначается ключевым словом DECLARE, а каждое объявление завершается символом точки с запятой (;).

После объявления переменных следует ключевое слово BEGIN, обозначающее начало основного программного блока. За ключевым словом BEGIN находятся команды, входящие в блок. Конец программного блока обозначается ключевым словом END.

#### **Пример 7.1:**

Создание функции на языке PL/pgSQL:

```
CREATE OR REPLACE FUNCTION a_function (i int4) RETURNS  
int4 AS  
$BODY$  
DECLARE  
    an_integer int4;  
    -- Объявление целочисленной константы.  
    -- инициализированной значением 5.  
    five CONSTANT integer := 5;  
    -- Объявление целочисленной переменной.  
    -- инициализированной значением 10.  
    -- Переменной не может присваиваться NULL,  
    ten integer NOT NULL := 10;  
BEGIN  
    an_integer := five * ten * i;  
    return an_integer;  
END;  
$BODY$  
LANGUAGE 'plpgsql';
```

Для того чтобы вызвать функцию необходимо записать ее имя, а также аргументы в секции FROM оператора SELECT.

#### **Пример 7.2:**

Вызов функции PL/pgSQL:

```
SELECT * FROM a_function (4);
```

### **Типы данных**

Переменные PL/pgSQL могут относиться к любому из стандартных типов данных SQL (например, integer или char). Помимо типов данных SQL, в PL/pgSQL также предусмотрен дополнительный тип RECORD, предназначенный для хранения записей без указания полей — эта информация передается при сохранении данных в переменной.



## Использование SELECT INTO

Команда SELECT INTO в основном требуется для сохранения данных записей в переменных, объявленных с типами %ROWTYPE и RECORD. Чтобы команда SELECT INTO могла использоваться с обычной переменной, тип этой переменной должен соответствовать типу поля, упоминаемому в команде SQL SELECT. Синтаксис команды SELECT INTO:

```
CREATE FUNCTION идентификатор (аргументы) RETURNS тип AS '  
DECLARE  
команда;  
BEGIN  
    SELECT INTO переменная [. ...] поле [. ...] секции_select;  
END;  
' LANGUAGE 'plpgsql';
```

В этом описании переменная — имя переменной, участвующей в присваивании, а секции\_select — любые поддерживаемые секции команды SQL SELECT, обычно следующие за списком целевых полей в команде SELECT.

Чтобы узнать, успешно ли были присвоены значения переменным командой SELECT INTO, воспользуйтесь специальной логической переменной FOUND:

```
IF NOT FOUND THEN      -- если не присвоены, то  
    [...]              -- делать  
END IF;
```

## Использование атрибута %TYPE

Атрибут %TYPE используется при объявлении переменных с типом данных, совпадающих с типом некоторого объекта базы данных (чаще всего поля). Синтаксис объявления переменной с атрибутом %TYPE:

```
переменная таблица. поле%TYPE
```

Это позволяет нам не привязываться к конкретному типу поля и еще более абстрагироваться от структуры хранения данных.

## Использование атрибута %ROWTYPE

Атрибут %ROWTYPE используется в PL/pgSQL для переменной-записи, имеющей одинаковую структуру с записями заданной таблицы. Не путайте атрибут %ROWTYPE с типом данных RECORD — переменная с атрибутом %ROWTYPE точно воспроизводит структуру туру записи конкретной таблицы, а переменная RECORD не структурирована и ей можно присвоить запись любой таблицы.

Работа с атрибутом %ROWTYPE отражается в примере 7.3. Для ссылки на имя поля в полученной переменной типа **authors%ROWTYPE** используется точка (.) по аналогии с составными структурами в языках программирования (Pascal, C).

### Пример 7.3:

Использование %ROWTYPE PL/pgSQL:

```
CREATE FUNCTION get_author (integer) RETURNS text AS '  
DECLARE  
-- Объявление псевдонима для аргумента функции.  
-- в котором должен передаваться код автора.  
author_id ALIAS FOR $1;  
-- Объявление переменной, структура которой  
-- совпадает со структурой таблицы authors.  
found_author authors%ROWTYPE;  
BEGIN  
-- Найти в таблице authors фамилию автора.  
-- код которого совпадает с переданным аргументом.  
SELECT INTO found_author * FROM authors WHERE id =  
author_id;  
-- Вернуть имя и фамилию, разделенные пробелом.  
RETURN found_author.first_name || " " || found_author.last_name;  
END;  
' LANGUAGE 'plpgsql';
```

### Передача управления

Команды передачи управления существуют практически во всех современных языках программирования, и PL/pgSQL не является исключением. С технической точки зрения сам вызов функции можно рассматривать как передачу управления последовательности команд PL/pgSQL. Тем не менее существуют и другие, более совершенные средства, определяющие последовательность выполнения команд PL/pgSQL. Здесь речь пойдет об условных командах IF/THEN и циклах.

Команда IF/THEN/ELSE задает команду (или блок команд), выполняемых в случае истинности некоторого условия. Синтаксис команды IF/THEN/ELSE:

```
CREATE FUNCTION функция (.аргументы) RETURNS тип AS '  
DECLARE  
объявления  
BEGIN  
        IF условие THEN  
                Команда;  
                [...]   
        /ELSE  
                Команда;  
                [...]   
        /END IF;  
END;  
LANGUAGE 'plpgsql';
```

Квадратные скобки, обрамляющие секцию ELSE, означают, что эта секция может быть опущена.

## Циклы

Самый простой вид цикла, используемого в PL/pgSQL, – безусловный. Его синтаксис:

```
LOOP
  команда ;
[...]
END LOOP;
```

Для выхода из цикла служит команда EXIT. Она также может содержать метку или условие завершения:

```
[ «метка» ]
LOOP
  команда;
[...]
EXIT [ метка ] [ WHEN условие ];
END LOOP;
```

Цикл WHILE выполняет блок команд до тех пор, пока заданное условие не станет ложным. Синтаксис использования следующий:

```
[ «метка» ]
WHILE условие LOOP
  команда;
[...]
END LOOP;
```

Перейдем к рассмотрению циклов FOR. Возможно, циклы FOR — самая важная разновидность циклов, реализованных в PL/pgSQL. Цикл FOR выполняет программный блок для целых чисел из заданного интервала. Ниже будет описан синтаксис цикла FOR.

```
[ «метка» ]
FOR переменная IN [ REVERSE ] выражение1 .. выражение2 LOOP
  команда;
END LOOP;
```

Цикл FOR выполняет одну итерацию для каждого значения переменной переменная в интервале, границы которого определяются выражениями выражение1 и выражение2 (включительно). Управляющую переменную цикла не обязательно объявлять вне блока FOR, если вы не собираетесь работать с ней после завершения цикла.

Циклы FOR также могут использоваться для перебора результатов запросов – ниже представлен соответствующий пример 7.4. Обратите внимание, что в примере используются не двойные кавычки, а дважды одинарные, как способ экранирования, т.к. тело функции само заключено в одинарные кавычки.

#### **Пример 7.4:**

Использование цикла FOR в PL/pgSQL:

```
CREATE FUNCTION extract_all_titles2 () RETURNS text AS '  
DECLARE  
-- Объявление переменной для кода темы.  
sub_id integer;  
-- Объявление переменной для хранения списка названий книг  
text_output text = '';  
-- Объявление переменной для названия темы.  
sub_title text;  
-- Объявление переменной для хранения записей.  
-- полученных при выборке из таблицы books.  
rowdata_books%ROWTYPE;  
BEGIN  
-- Внешний цикл FOR: тело цикла выполняется до тех пор.  
-- пока переменная i не станет равна 15. Перебор начинается с 0.  
-- Следовательно, тело цикла будет выполнено 16 раз  
-- (по одному для каждой темы).  
FOR i IN 0..15 LOOP  
-- Получить из таблицы subjects название темы.  
-- код которой совпадает со значением переменной i.  
SELECT INTO subtitle subject FROM subjects WHERE id = i;  
-- Присоединить название темы, двоеточие и символ новой строки  
-- к переменной text_output.  
text_output = text_output || '\n' || sub_title || ':\n';  
-- Перебрать все записи таблицы books.  
-- у которых код темы совпадает со значением переменной i.  
FOR row_data IN SELECT * FROM books WHERE subjected = i  
LOOP  
-- Присоединить к переменной text_output название книги  
-- и символ новой строки.  
text_output := text_output || row_data.title || '\n';  
END LOOP;  
END LOOP;  
-- Вернуть список.  
RETURN text_output;  
END;  
' LANGUAGE 'plpgsql';
```

#### **Обработка ошибок и исключений**

Команда RAISE предназначена для инициирования ошибок и исключений в функциях PL/pgSQL. Она передает заданную информацию механизму PostgreSQL elog. В команде RAISE также указывается уровень ошибки и строка, передаваемая PostgreSQL. Кроме того, в команду можно

включить переменные и выражения, значения которых будут содержаться в выходных данных. Соответствующие позиции строки помечаются знаками процента (%). Синтаксис команды RAISE:

*RAISE уровень "сообщение" [. идентификатор [...] ];*

Уровень ошибки может принимать следующие значения: DEBUG, NOTICE, EXCEPTION.

Для того, чтобы обработать исключение, возникшее в текущей транзакции (т.е. между BEGIN и END) необходимо записать следующую конструкцию:

*EXCEPTION*

*WHEN код\_исключения THEN*

*команда;*

*[WHEN код\_исключения THEN*

*команда;]*

На следующем примере 7.5 представлена функция, в теле которой обрабатываются два вида исключений: деление на ноль и взятие корня от отрицательного числа. Попробуйте создать эту функцию, а затем вызывайте ее с параметрами: 1, 0, -1.

#### **Пример 7.5:**

Демонстрация обработки исключений:

```
CREATE OR REPLACE FUNCTION tt(ii integer) RETURNS boolean  
AS $BODY$  
  DECLARE  
    i integer := 100;  
    result bool := true;  
  BEGIN  
    i:=i/ii;  
    i:=sqrt(i);  
    return result;  
  EXCEPTION  
  WHEN DIVISION_BY_ZERO THEN  
    result := false;  
    return result;  
  WHEN INVALID_ARGUMENT_FOR_POWER_FUNCTION THEN  
    result := false;  
    return result;  
  END;  
$BODY$  
LANGUAGE 'plpgsql' VOLATILE;
```

Таблицу констант (кодов) ошибок, возникающих в PostgreSQL, можно посмотреть в документации. Для этого запустите PgAdmin, запустите редактор SQL-запросов. В редакторе выберите пункт меню «Help»-«Help». Затем перейдите не раздел «Appendix A. PostgreSQL Error Codes».

## Вызов функций в теле процедуры

При вызове функции PL/pgSQL из кода PL/pgSQL имя функции обычно включается в команду SQL SELECT или в команду присваивания:

```
SELECT функция (.аргументы);
```

```
переменная := функция(аргументы);
```

Подобный способ вызова функций при выборке и присваивании стал стандартным, поскольку любая функция PostgreSQL должна возвращать значение некоторого типа. Ключевое слово PERFORM позволяет вызвать функцию и проигнорировать возвращаемое значение:

```
PERFORM функция (аргументы);
```

## PL/pgSQL и триггеры

Определения триггеров PostgreSQL могут содержать ссылки на триггерные функции (то есть функции, которые должны вызываться при срабатывании триггера), написанные на языке PL/pgSQL. Триггер определяет операцию, которая должна выполняться при наступлении некоторого события в базе данных.

Не путайте определение триггера с определением триггерной функции. Триггер определяется командой SQL CREATE TRIGGER, а триггерная функция определяется командой SQL CREATE FUNCTION. Синтаксис определения триггера следующий:

```
CREATE FUNCTION функция () RETURNS opaque AS '
```

```
DECLARE
```

```
    объявления;
```

```
BEGIN
```

```
    команды;
```

```
END;
```

```
' LANGUAGE 'plpgsql';
```

Ниже приведен пример создания триггерной функции. Обратите внимание, что *opaque* заменено на “*trigger*”, что допустимо.

### Пример 7.6:

Триггерная функция, которая автоматически для вставляемого значения поля считает его md5 сумму:

```
CREATE OR REPLACE FUNCTION hashpassword()
```

```
RETURNS "trigger" AS
```

```
$BODY$
```

```
BEGIN
```

```
    NEW."password"=md5(NEW."password");
```

```
    RETURN NEW;
```

```
END
```

```
$BODY$
```

```
LANGUAGE 'plpgsql' VOLATILE;
```

```
ALTER FUNCTION "UMUAccessManagement".hashpassword()  
OWNER TO postgres;
```

В таблице 7.1 приведены специальные переменные, которые могут быть использованы в триггерных функциях.

Таблица 7.1 – Специальные переменные в триггерных функциях

Имя	Тип данных	Описание
NEW	RECORD	Новая запись базы данных, созданная командой INSERT или UPDATE при срабатывании триггера уровня записи (ROW). Переменная используется для модификации новых записей
OLD	RECORD	Старая запись базы данных, оставшаяся после выполнения команды INSERT или UPDATE при срабатывании триггера уровня записи (ROW)
TG_NAME	name	Имя сработавшего триггера
TG_OP	text	Строка INSERT, UPDATE, DELETE в зависимости от операции
TG_RELNAME	name	Имя таблицы, в которой сработал триггер

## Задание на лабораторную работу №7

Данная лабораторная работа не является обязательной для выполнения, однако необходима для получения оценки выше «4» на экзамене.

1. Создайте отношение А. Для этого отношения определите два поля: поле – номер (тип число) и поле – строка\_значений (тип одномерная матрица). Таблица будет иметь следующую структуру:

номер_матрицы	строка_значений
1	'{2,4}'
1	'{-3,1}'
2	'{0}'
2	'{5}'
3	'{-2,-4}'

Что соответствует матрицам  $A1 = \begin{pmatrix} 2 & 4 \\ -3 & 1 \end{pmatrix}$ ,  $A2 = \begin{pmatrix} 0 \\ 5 \end{pmatrix}$  и  $A3 = \begin{pmatrix} -2 & -4 \end{pmatrix}$ .

2. Создайте отношение В, содержащее четыре поля: номер\_операции (целое), номер\_первой матрицы (целое), номер\_второй матрицы (целое), название\_функции (текст).

3. Создайте третье отношение – С, которое имеет поля: номер\_операции (целое) и поле результат (numeric).

4. Создайте функции сложения, вычитания, транспонирования, умножения матриц, которые работают с матрицами, определенными в отношении А, т.е. на вход получают номер матрицы (или матриц – для бинарных операций).

5. Создайте функцию вычисления определителя матрицы по ее номеру.

6. Создайте триггер, который:

- при добавлении новой строки в отношение В (с новым номером\_операции) производит расчет для этой операции, вызвав соответствующую функцию для переданных матриц, и записывает полученный результат в отношение С в виде нового кортежа;
- при обновлении строки в отношении В производит пересчет, согласно новых переданных параметров, если пересчет произведен без ошибок, то обновляет соответствующий кортеж в отношении С;
- при выборе строки из отношения В производит пересчет, согласно существующих параметров.

### Прием работы

Прием происходит при наличии оформленного отчета и работающей БД.

### Вопросы

1. Чем отличается использование атрибута %ROWTYPE от типа RECORD?
2. Что такое PL/pgSQL и из каких блоков состоит процедура на этом языке?
3. Что такое триггеры и триггерные функции?
4. Как можно вставить данные в переменную типа RECORD?
5. Какие циклы существуют в языке PL/pgSQL.



## Лабораторная работа №8

**Тема:** Объектно-ориентированный подход в создании БД с использованием объектно-реляционной СУБД PostgreSQL.

**Цель:** моделирование объектного подхода на реляционной БД.

**Навыки и умения:** написание хранимых процедур на языке PL/pgSQL, создание представлений, написание триггеров и триггерных функций, написание агрегатных функций.

### Теоретический базис

#### Создание агрегатных функций

СУБД PostgreSQL позволяет создавать собственные агрегатные функции для применения их в запросах группировки. Для определения новой агрегатной функции в базе данных служит конструкция CREATE AGGREGATE. Синтаксис конструкции следующий:

```
CREATE AGGREGATE имя ( BASETYPE = входной_тип  
[ , SFUNC - функция. STYPE = переходный_тип ]  
[ , FINALFUNC - завершающая_функция ]  
[ , INITCOND - начальное_состояние ] )
```

Расшифровка параметров:

- имя. Имя создаваемой агрегатной функции;
- входной\_тип. Тип входных данных, с которыми работает создаваемая функция. Если агрегатная функция игнорирует входные данные (как, например, функция count()), вместо типа данных указывается строковая константа ANY;
- функция. Имя функции, вызываемой для обработки всех входных данных, отличных от NULL. Обычно такая функция получает два аргумента: первый аргумент относится к типу данных переходный\_тип, а второй — к типу данных входной\_тип. Если агрегатная функция не анализирует входные данные, она получает только один аргумент типа переходный\_тип. Так или иначе, функция должна возвращать значение типа переходный\_тип;
- переходный\_тип. Промежуточный тип данных агрегатной функции;
- завершающая\_функция. Имя итоговой функции, вызываемой для вычисления результата агрегатной функции после обработки всех входных данных. Функция должна получать один аргумент типа переходный\_тип. Выходной тип данных агрегатной функции определяется типом возвращаемого значения этой функции. Если параметр FINALFUNC не указан, то последнее переходное значение передается в качестве результата агрегатной функции, а выходной тип данных определяется типом переходный\_тип;

- `начальное_состояние`. Начальное состояние промежуточного значения агрегатной функции. Задается литералом типа `переходный_тип`. Если параметр `начальное_состояние` не задать, промежуточное значение инициализируется псевдозначением `NULL`.

Агрегатные функции характеризуются в первую очередь типом входных данных. Допускается существование двух и более агрегатных функции с одинаковыми именами, вызываемых с разными типами данных (это называется перегрузкой функций).

#### **Пример 8.1:**

В следующем примере определяется агрегатная функция `sum()`, работающая с текстовыми данными. Она вызывает встроенную функцию PostgreSQL – `textcat(text, text)` для конкатенации всего текста во входных данных:

```
CREATE AGGREGATE sum ( BASETYPE = text,  
    SFUNC = textcat,  
    STYPE = text,  
    INITCOND = " );
```

Для проверки работы созданной в примере 8.1 функции-агрегата выполните запрос и разберитесь, что он делает:

```
SELECT substring(t2.r,1,length(t2.r)-1)  
FROM (SELECT sum(t.title || ',') as r  
FROM (SELECT * FROM (SELECT 'world' as title union SELECT 'Hello')  
as t1 order by t1.title) as t) as t2
```

## Задание на лабораторную работу №8

Данная лабораторная работа не является обязательной для выполнения, однако необходима для получения оценки выше «4» на экзамене.

### Необходимо:

1. Создать несколько отношений, связанных в виде иерархии, как это показано на рис. 8.1:

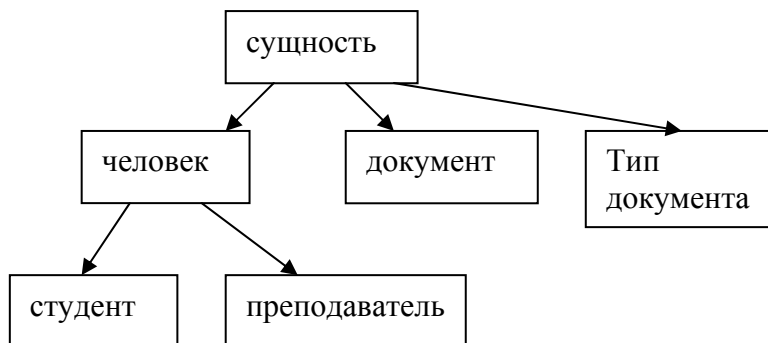


Рисунок 8.1 – Структура объектно-реляционных связей

2. Самостоятельно определить атрибуты этих отношений.
3. Иерархию реализовывать с использованием наследования (л/р 6).
4. Создать представление, которое выбирает все атрибуты объекта и его наследников в один кортеж. В случае, если для какого-то из атрибутов имеется несколько значений необходимо формировать поле в следующем виде:  
{<1ое значение атрибута>, <2ое значение атрибута>, ...}, где <i-ое значение атрибута> - значение атрибута в i-ом кортеже для объекта. Для этого написать собственную агрегатную функцию(ии), работающую с типами integer, text, timestamp.
5. Определить универсальные функции для удаления, добавления, обновления любого объекта, которым передается имя отношения, фильтр (если нужно), массив имен полей (если нужно), массив новых значений полей (если нужно).
6. На основании функций из п.5 определить для каждого объекта БД (кроме «сущность») функции: добавить, изменить, удалить. В функциях должен быть реализован контроль за уникальностью объекта.
7. Запретить добавление данных в отношения с использованием SQL запросов (т.е. не через интерфейсные функции из пункта 5). Для этого определить необходимые триггеры.
8. Написать функции (PL/PGSQL) вывода существующих документов человека (например, паспорт, з/к – если студент, № пропуска – если преподаватель). Функция использует представление созданное в пункте 4. Не использовать внешние ключи (реляционные связи) для связывания отношений находящихся в одной ветке иерархии, но использовать их (если

необходимо) для связи объектов на одном уровне иерархии. Значение потенциального ключа в базовых таблицах не должно повторяться даже при выполнении запроса без параметра ONLY.

9. Структура БД, ограничения, правила наследования, процедуры, представления, а также данные должны быть представлены в виде SQL-скрипта.

## **Список используемой литературы**

1. Дейт К.Дж. Введение в системы баз данных. М.: Вильямс, 2000.
2. Харрингтон Дж. Проектирование объектно-ориентированных баз данных. М.: Вильямс, 2000.
3. Грабер М. SQL Справочное руководство. 2-е изд. – М.: Издательство «Лори», 2001.
4. Астахова И.Ф. SQL в примерах и задачах: Учеб. пособие / И.Ф. Астахова, А.П. Толстобров, В.М. Мельников. – Мн.: Новое знание, 2002.
5. Вейскас Дж. Эффективная работа с Microsoft Access 2000. - СПб.: Питер, 2000.
6. Уорсли Дж., Дрейк Дж. PostgreSQL. Для профессионалов. - СПб.: Питер, 2003.