Язык запросов SQL

Для доступа прикладного программного обеспечения к информации, хранимой в базе данных, существует множество более или менее специализированных языков. Такие, как dBase (и все его семейство – Clipper, FoxPro), Access, Paradox являются одновременно и языками программирования (или содержат средства доступа к таковым), что позволяет писать достаточно сложные прикладные программы. Их достоинство в том, что язык доступа к базе данных является частью самого языка программирования – они неотделимы друг от друга и органично дополняют друг друга, позволяя сделать многие интересные вещи очень простым путем (типа вычисляемого в процессе выполнения программы выражения макрокомпилятора Clipper). Недостаток же является прямым продолжением достоинства – подобная интегрированность является узким местом при малейшей попытке смены окружения прикладной программы, да и организация связей с модулями, написанными на других языках была очень сложна, если возможна вообще – программисты на Clipper помнят, скольких усилий требовалось для сопряжения хотя бы с языком Си, имеющим принципиально открытую архитектуру. Другой недостаток проистекает из «темного прошлого» такого рода языков. Поскольку они возникали как часть СУБД конкретной, то исполнялись лишь под управлением «своей» СУБД в режиме интерпретации. В отсутствие последней работа прикладной программы была невозможна. При переходе к компилирующим трансляторам разработчики СУБД шли на различные ухищрения, со временем переставшие удовлетворять программистов.

Наконец, были написаны библиотеки API, позволяющие осуществлять доступ к БД из языков программирования общего назначения (Paradox API для Си, библиотеки доступа к dBase-совместимым таблицам практически под все языки вплоть до Бейсика и т.д.) Все это преследовало лишь одну цель – разделить язык программирования и язык манипулирования данными. Т.е., программист писал обращения на языке манипулирования данными, которые обрабатывала библиотека доступа к данным.

Язык SQL был создан фирмой IBM в 70-х годах как язык, посредством которого конечные пользователи могли бы самостоятельно писать запросы к базе данных (в ту пору квалификация пользователей еще позволяла возложить на последних такую обязанность). Первоначальная версия называлась SEQueL (**S**implified **E**nglish **Que**ry **L**anguage), после некоторых доработок появился язык SQL (Structured Query Language – структурированный язык запросов) в

его теперешнем виде. Фирма IBM тогда еще вполне могла диктовать моду, поэтому SQL со временем стал промышленным стандартом, и многие СУБД в той или иной мере стали поддерживать обработку SQL-запросов.

С появлением механизма ODBC (**O**pen **D**ata**B**ase **C**onnection) от компании Microsoft продвижение языка SQL как промышленного стандарта заметно ускорилось. Как правило, конкретные ODBC-драйвера поддерживают SQL в той или иной мере независимо от того, каков реальный источник данных. OBDC механизм позволяет обращаться как к реальным базам данных, так и просто к таблицам, хранимым на локальной машине. Фирма Borland (ныне Inprise) создала свой собственный механизм под названием BDE (Borland Database Engine), но он прижился гораздо хуже и в настоящее время BDE пользуется тем же самым ODBC, поддерживая его в полной мере. Об удачности ODBC-механизма говорит хотя бы и тот факт, что ODBC применяется даже в мире Unix-подобных систем, где традиционно более чем скептически относятся ко всему, что изобретено в Ричмонде.

Язык SQL не является языком программирования как таковым – это язык, ориентированный на работу с множествами. С помощью этого языка можно сформировать очень сложный запрос, но достаточно трудно воплотить какой-либо ветвистый алгоритм – SQL просто не предназначен для этого.

Основным предложением языка является инструкция SELECT (выбор), производящая выборку информации из базы данных по указанным критериям. Синтаксис будет рассмотрен ниже.

Структура сетевой СУБД

На рисунке видно, что в сети база данных, находящаяся на сервере, удалена от клиента, чего не было ранее, когда либо имела место просто однопользовательская структура, либо пользователи сидели за терминалами большой ЭВМ, что практически одно и то же с точки зрения самой СУБД – процесс клиента и процесс СУБД выполняются на одном компьютере и могут взаимодействовать в любой момент, когда им это необходимо. При работе же в сети структура СУБД стала разнесенной – появилась серверная часть, управляющая самой БД, хранимой на носителе сервера и часть клиентская, желающая получать информацию с сервера и сохранять ее там. Не следует считать, что вся СУБД расположена только на сервере либо же только на клиентской станции – СУБД разделена и состоит из двух частей, работающих в постоянном взаимодействии.

Очевидно, что объем передаваемой по сети информации должен быть максимально минимизирован, ибо пропуская способность сетей, даже локальных, весьма низка по сравнению с пропускной способностью внутренних магистралей ЭВМ. Данному условию не удовлетворяет организация вычислительной системы с применением файл-сервера, ибо в

таком случае обработку собственно запроса выполняет клиентская станция, что крайне резко увеличивает объем передаваемых по сети данных. Собственно, при такой архитектуре (что используется некоторыми современными программными продуктами типа «сетевой» версии 1С: Бухгалтерия) система управления базой данных распложена на клиентской машине. Сервер предоставляет только услуги по хранению файлов, т.е. нижнего уровня структуры СУБД – все это крайне отрицательно сказывается на надежности и скорости системы в целом.

MS SQL Server

MS SQL Server предназначен для управления реляционными базами данных. Соединение клиента с сервером может быть организовано как с использованием любого сетевого протокола (TCP/IP, IPX/SPX, NetBEUI), так и с помощью именованных каналов (Named Pipes) Windows NT. Сервер функционирует под управлением ОС семейства Microsoft Windows NT (Windows'2000, XP), но в локальном варианте может быть установлен и под Windows'9x (Windows Millennium). Клиент может быть расположен как на том же компьютере, что и сервер, так и на любом другом компьютере сети. Все действия над сервером кроме установки могут быть проделаны с любой клиентской машины. Запуск службы сервера при использовании Windows NT ('2000, XP) может также осуществляться с клиентской машины.

Типы данных SQL

Всякое поле таблицы, всякая локальная переменная, выражение и параметр процедуры имеют тип. Стандартные типы данных перечислены ниже.

Целочисленные

Bit – целое, имеющее значение 0 либо 1.

Int – целое число от -2^{31} (-2,147,483,648) до $2^{31}-1$ (2,147,483,647).

Smallint – целое число от -2^{15} (-32,768) до $2^{15}-1$ (32,767).

Tinyint – целое число от 0 до 255.

Десятичные и числовые

Decimal – число с фиксированной точностью в диапазоне от $-10^{38}-1$ до $10^{38}-1$.

Numeric – синоним к «Decimal».

Денежные

Money – денежная величина от -2^{63} (-922,337,203,685,477.5808) до $2^{63}-1$ (+922,337,203,685,477.5807), с точностью до десятитысячной части денежной единицы.

Smallmoney – денежная величина от -214,748.3648 до +214,748.3647, с точностью до десятитысячной части денежной единицы.

Приближенные числовые

Float – число с плавающей точкой от -1.79E+308 до 1.79E+308.

Real – действительное число с плавающей точкой -3.40E+38 до 3.40E+38.

Дата и время

Datetime – дата и время с 1 января 1753 года до 31 декабря 9999 года, с точностью до 3.33 миллисекунд.

Smalldatetime – дата и время с 1 января 1900 года до 6 июня 2079 года, с точностью до минуты.

Перечислимые

Cursor – ссылка на курсор (выборка данных с сервера).

Timestamp – уникальный в рамках базы данных идентификатор.

Uniqueidentifier – глобальный уникальный идентификатор.

Символьные строки не unicode-формата

Char – строка фиксированной длины размером до 8000 символов.

Varchar – строка переменной длины размером до 8000 символов.

Text – строка переменной длины размером до $2^{31}-1$ (2,147,483,647) символов.

Символьные строки unicode-формата

Nchar – строка фиксированной длины размером до 4000 символов.

Nvarchar – строка переменной длины размером до 4000 символов.

Ntext – строка переменной длины размером до $2^{30}-1$ (1,073,741,823) символов.

Двоичные строки

Binary – двоичное данное фиксированной длины объемом до 8000 байт.

Varbinary – двоичное данное переменной длины объемом до 8000 байт.

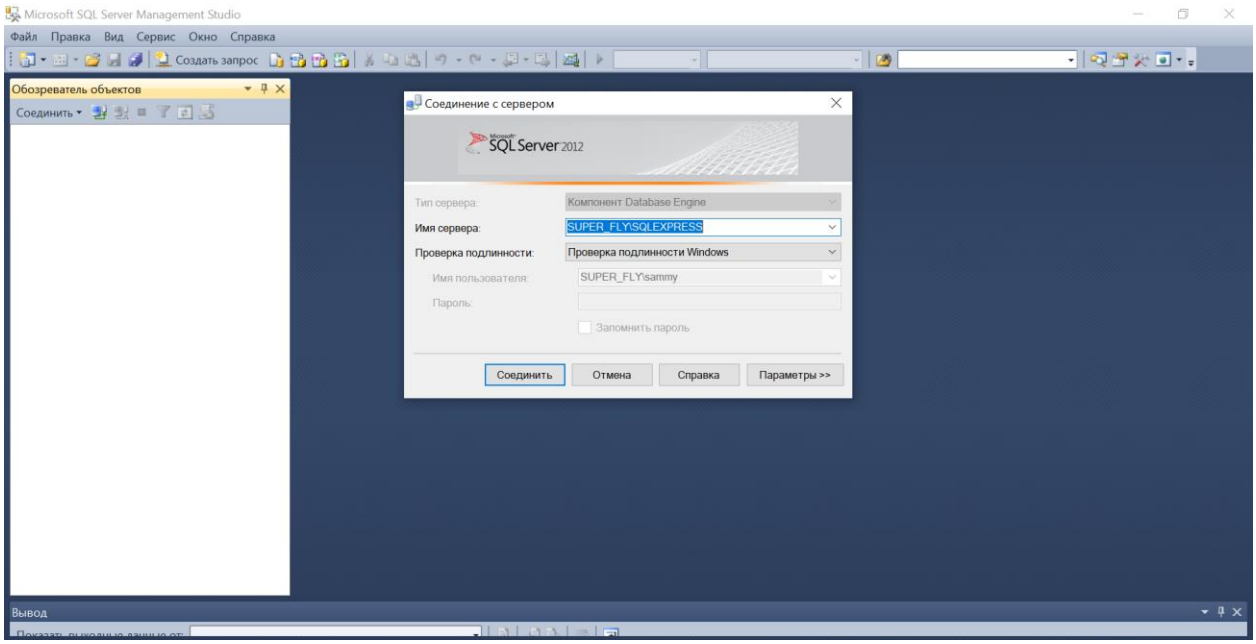
Image – двоичное данное переменной длины объемом до 2^{31}-1 (2,147,483,647) байт.

Кроме того, существует пустой тип, не связанный ни с каким другим. Пустой тип означает отсутствие данных в поле (переменной). Это не ноль для числового поля и не пустая строка для текстового – это именно отсутствие данных.

Задание: База данных авторов и написанных ими книг. Будем полагать, что у каждой книги автор один, но один и тот же автор мог написать несколько книг, издаваемых в разных издательствах.

Выполнение:

Запускаем ***Среда SQL Server Management Studio***, и создаём, согласно варианту задания, структуру данных указанного назначения. Ниже на рисунке 1, представлены таблицы базы данных и Главное окно программы:



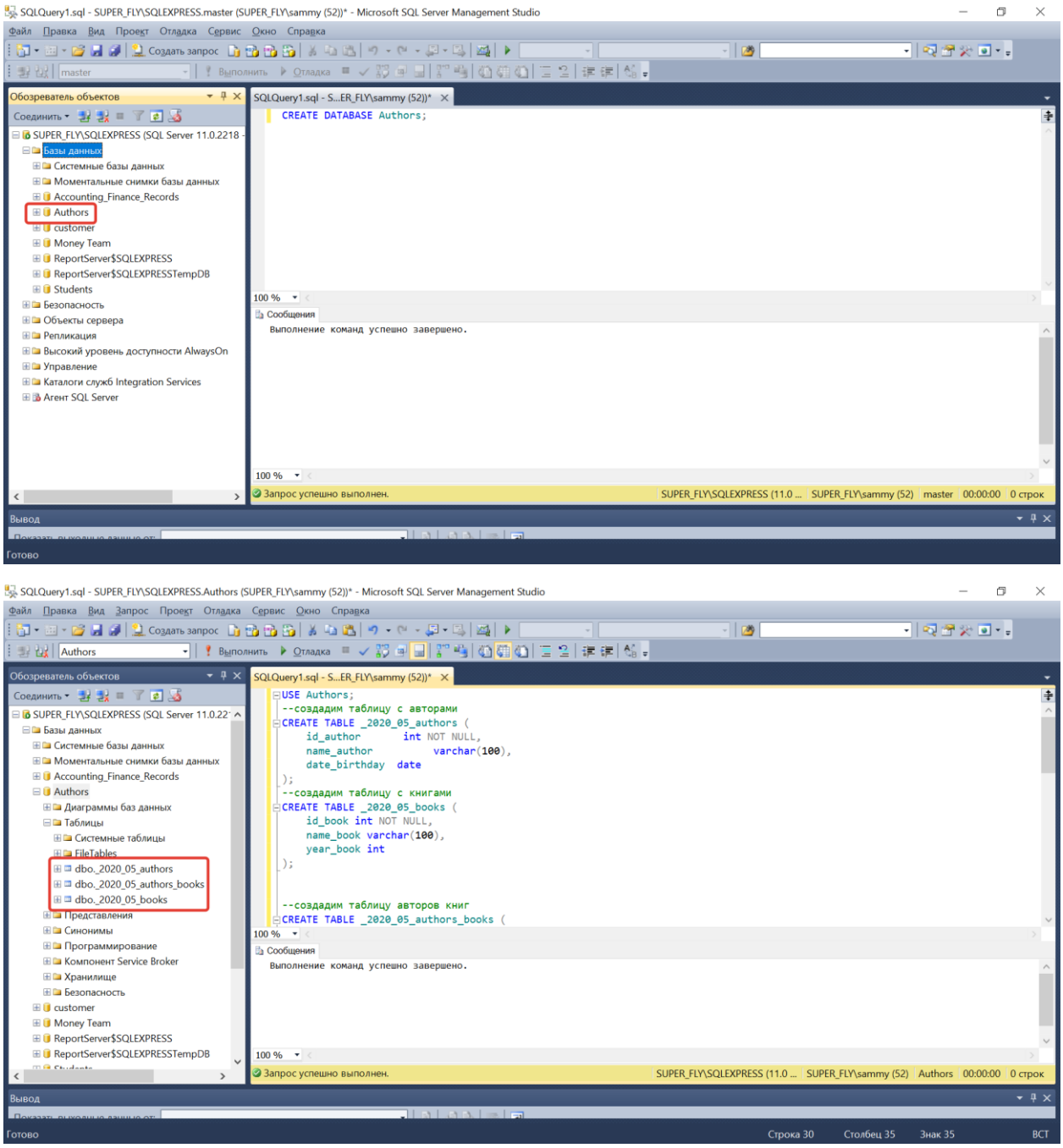


Рисунок 1. – Главное окно программы MSSQL.

Далее представлен текст запроса для создания структуры данных указанного назначения.

Текст запроса:

```
USE Authors;
--создадим таблицу с авторами
CREATE TABLE _2020_05_authors (
    id_author       int NOT NULL,
      name_author           varchar(100),
      date_birthday  date
);
```

```
--создадим таблицу с книгами
CREATE TABLE _2020_05_books (
	id_book int NOT NULL,
	name_book varchar(100),
	year_book int
);


--создадим таблицу авторов книг
CREATE TABLE _2020_05_authors_books (
	id_authors_books int NOT NULL,
	id_author int,
	id_book int
);
--Установим первичные ключи
ALTER TABLE _2020_05_authors
ADD PRIMARY KEY(id_author)

ALTER TABLE _2020_05_books
ADD PRIMARY KEY (id_book)

ALTER TABLE _2020_05_authors_books
ADD PRIMARY KEY (id_authors_books)

--Установим внешние ключи с каскадным удалением

ALTER TABLE _2020_05_authors_books
ADD
FOREIGN KEY (id_author) REFERENCES _2020_05_authors (id_author)
ON DELETE CASCADE

ALTER TABLE _2020_05_authors_books
ADD
FOREIGN KEY (id_book) REFERENCES _2020_05_books (id_book)
ON DELETE CASCADE
```

Ниже на рисунке 2, приведена диаграмма базы данных.

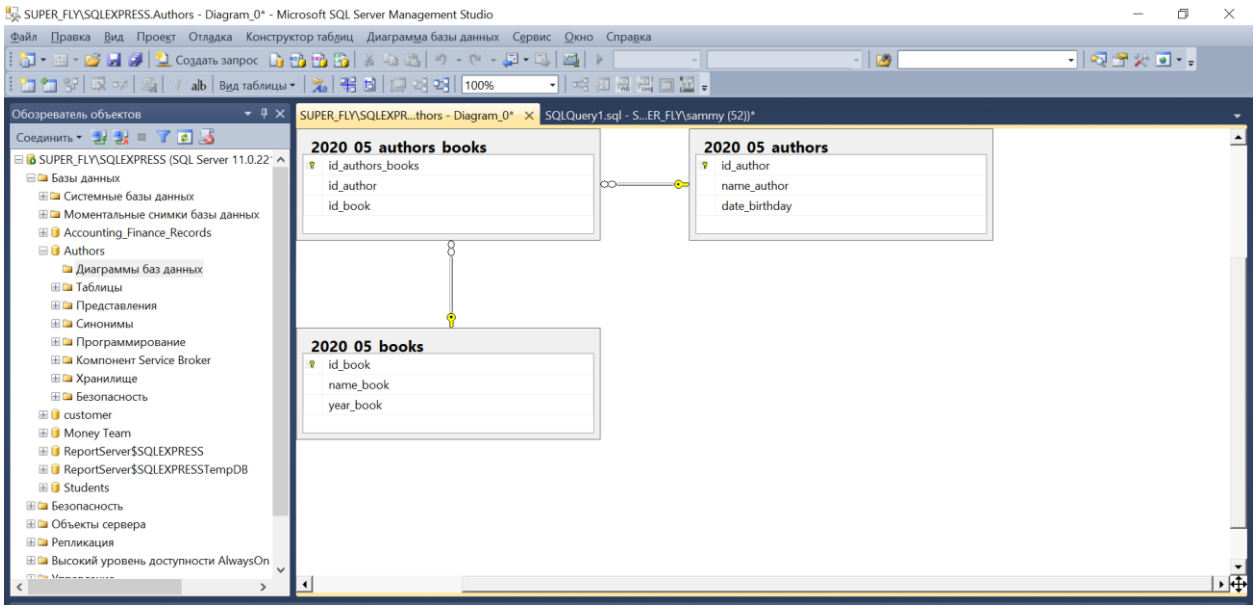


Рисунок 2. – Диаграмма базы данных.

Вывод:

В данной работе, была разработана структура данных, согласно варианту задания, а также были приведены таблицы до третьей нормальной формы.

SQL-запросы и представления. Организация транзакций.

Любое взаимодействие с SQL-сервером производится посредством запросов, даже если это явно не показано. Например, программа Enterprise Manager, известная вам по прошлой работе, взаимодействует с сервером именно таким образом. По существу, она, да и все прочие приложения являются лишь локальным интерфейсом при клиент-серверном взаимодействии.

В данной работе вам предстоит познакомиться с непосредственным взаимодействием с сервером посредством программы SQL Query Analyzer. Кроме того, будет рассмотрено создание индексов таблиц.

Все запросы, как уже было сказано, проводятся на языке SQL. В данной работе будут рассмотрены не все аспекты взаимодействия, а только добавление данных в таблицу, изменение данных, удаление записей и выборка данных.

Синтаксис предложений SQL

Всякое предложение SQL состоит из ключевого слова и параметров, определяющих функционирование сервера при его выполнении. SQL содержит значительное количество команд, позволяющих в полной мере управлять как самим сервером, так и хранимыми на нем данными и всем, что к ним относится (метаданные, пользователи и их права). Мы рассмотрим лишь основные команды, без которых невозможна работа с сервером.

Следует отметить, что SQL сервер позволяет использовать пробелы и зарезервированные слова в идентификаторах. Для этого идентификаторы должны быть заключены в квадратные скобки в конечном запросе. При описании синтаксиса квадратные скобки используются для указания необязательных конструкций. Известна проблема с идентификаторами, начинающимися на русскую букву «а».

INSERT

Предложение INSERT предназначено для добавления полей в таблицы базы данных.

INSERT INTO <таблица> [(<поле>[,...])

VALUES (DEFAULT | NULL | <выражение>[,...]) | DEFAULT VALUES

Указываются поля таблицы, подлежащие заполнению и значения в этих полях. Вместо указания значения можно указать предложение NULL (в поле будет вставлено пустое значение) либо DEFAULT (будет вставлено значение

по умолчанию, если оно задано в структуре таблицы). Здесь приведен один из вариантов данного предложения.

SELECT

Предложение SELECT предназначено для выборки данных. Кроме того, это предложение может использоваться для иных целей – например, в теле хранимых процедур присвоение значения переменной может быть осуществлено только с использованием данной инструкции. Синтаксис предложения SELECT приводится ниже. Следует отметить, что это отнюдь не полный перечень возможностей, а только часть.

SELECT [ALL | DISTINCT]
[{TOP n [PERCENT]]
<список_выборки>
[INTO <новая_таблица>]
[FROM {<источник>} [,...n]]
[WHERE <условие_отбора>]
[GROUP BY <условие_группировки> [,...n]
[ORDER BY <условие_сортировки> [ASC | DESC]]
Где <список_выборки> ::= {* | { <столбец> | <выражение>} [[AS] <алиас_столбца>]}

Параметры инструкции SELECT:

ALL означает, что следует вернуть все записи, отвечающие условиям выборки. Это поведение по умолчанию.

DISTINCT означает, что следует вернуть только различные строки запроса. Если в выборке есть одинаковые строки, лишние дубликаты будут удалены.

TOP n [PERCENT] указывает, что следует вернуть только первые n записей выборки. Если указано слово PERCENT, то будут возвращены только n процентов всей выборки.

***** (звездочка) означает, что следует вернуть все столбцы в том порядке, в каком они содержатся в источнике.

Источник – имя таблицы активной базы данных либо имя представления. Может быть указан алиас (псевдоним) таблицы. Источников может быть несколько.

Столбец в списке выборки может содержать и имя таблицы, чтобы избежать путаницы при обращении к одинаково названным столбцам разных таблиц. Например, *_users.ID* означает столбец *ID* таблицы *_users*. **Алиас**

столбца предназначен для переименования столбца в рамках данной выборки, с той же целью. Далее мы будем полагать, что везде, где требуется указать стобец, может быть использован его псевдоним.

Новая_таблица указывает имя новой таблицы, которая будет создана для помещения в нее результатов обработки запроса.

Условие_отбора представляет собой собственно инструмент формирования выборки из всего множества данных в данном источнике. Как правило, это оператор сравнения. Подробнее данный оператор рассмотрен в лекционном курсе. См. также список операторов в лабораторной работе №4 в этом же пособии.

Задание **условия_группировки** позволяет сгруппировать одинаковые строки выборки. Как правило, это имя столбца в выборке. Поля, имеющие типы *text, ntext, image, bit* не могут быть использованы.

Условие_сортировки, будучи задано, упорядочивает выборку по указанному выражению. Как правило, указывается столбец выборки. Можно указать несколько столбцов. Столбцы, имеющие тип *ntext, text, image* не могут быть указаны. Параметр **ASC** требует сортировать выборку в возрастающем (неубывающем) порядке, **DESC** в обратном. Пустые значения воспринимаются как имеющие минимальный вес.

Примеры:

SELECT * FROM _users

Выборка всей таблицы *_users*. Столбцы будут возвращены в том порядке, в каком они присутствуют в таблице.

SELECT [ID], [Name] FROM _users WHERE [Name] = ‘Oleg’

Выборка полей *ID, Name* из таблицы *_users* в той ее части, где поле *Name* имеет значение ‘Oleg’.

SELECT * FROM goods WHERE [Date] > 13.10.1999 ORDER BY [Name]

Выборка всех полей из таблицы *goods*. Выбираются записи, где значение поля *Date* содержит значения, большие 13 декабря 1999 года. Выборка упорядочивается по полю *Name*.

SELECT * FROM [студенты] WHERE [номер группы] = 4512 GROUP BY [рейтинг]

Из таблицы *студенты* выбираются все студенты, имеющие номер группы 4512 и выборка группируется по рейтингу.

Наряду с выборкой, с базой данных приходится производить и такие операции как добавление, удаление и изменение данных.

DELETE

DELETE FROM <источник> WHERE <условие_отбора>

Производится удаление записей из указанного источника (таблицы), условие отбора аналогично предложению SELECT.

Примеры:

DELETE FROM _users WHERE [ID] = 2387

Из таблицы *_users* удаляются строки, имеющие значение 2387 в поле *ID*.

DELETE FROM [студенты] WHERE [студенты].[имя] = (SELECT [имя], [оценка] FROM [экзамены].[оценка] = 2)

Из таблицы *студенты* удаляются записи о тех студентах, для которых *оценка* в таблице *экзамены* равна двойке.

Внимание! В случае исполнения конструкции типа «DELETE FROM _table» из указанной таблицы будут удалены все записи!

UPDATE

UPDATE <источник> SET <список_присвоений> WHERE <условие_отбора>

Производится изменение значений полей источника в соответствии со списком присвоений. Условие отбора то же.

Примеры:

UPDATE _users SET [Name] = ‘David’ WHERE [Name] = ‘Oleg’

Пользователь имя *Oleg* в таблице *_users* заменяется на *David*.

UPDATE [студенты] SET [имя] = ‘Иванова А.Б.’ WHERE [имя] = ‘Петрова А.Б.’

Студентка *Иванова А.Б.* в таблице *студенты* переименовывается в *Петрову*.

UPDATE [студенты] SET [на отчисление] = TRUE WHERE [имя] = (SELECT [имя], [оценка] FROM [экзамены] WHERE [оценка] = 2)

Для студентов, сдавших экзамен на двойку, проставляется флаг на отчисление (значение булевского поля *на отчисление* меняется на значение *Истина*).

Представления («Views»)

Предложение SELECT с указанием нескольких таблиц позволяет производить любую выборку данных. Вместе с тем, существует еще один механизм, позволяющий выбирать связанную информацию из нескольких таблиц одновременно. Предложение SQL SELECT позволяет задействовать несколько таблиц, увязав их в достаточно сложную конструкцию и получить необходимые данные в требуемой форме. Вместе с тем, сложная инструкция SELECT выполняется достаточно долго, ибо интерпретируется сервером «на ходу», причем интерпретируется заранее. Повысить скорость выполнения такого рода операций, одновременно улучшив их читабельность и упростив отладку, позволяет организация представлений.

Представление – это своего рода источник данных, могущий быть употребленным в инструкции SQL SELECT. Само представление выбирает данные из таблиц либо иных представлений. Поскольку представление хранится на сервере, последний имеет возможность предкомпиляции представления, что повышает скорость его обработки.

Задание 2. Вывод книг, написанных не ранее 1999 года. Должны возвращаться: Название книги, автор, год издания.

Выполнение:

Запускаем ***Среда SQL Server Management Studio***, и подключаемся к базе данных, созданную из предыдущей лабораторной работы. Ниже на рисунке 1, представлена база данных, таблицы ***Authors***, и Главное окно программы:

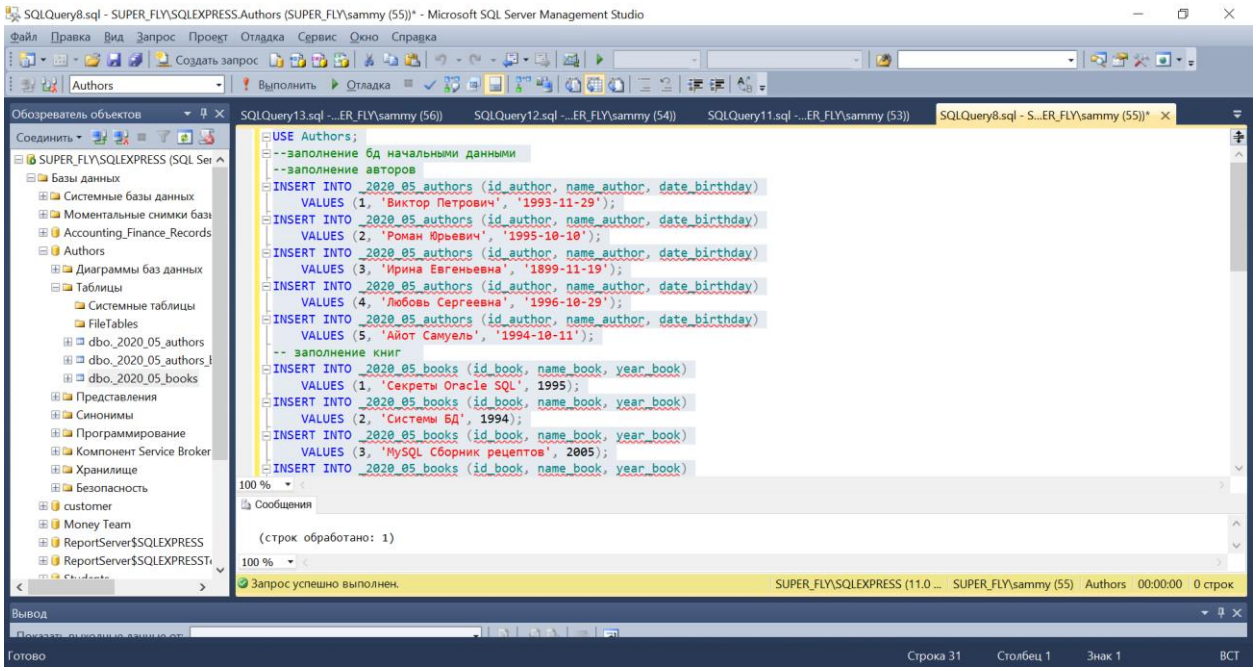


Рисунок 1. – Главное окно программы

Заполним базу данных начальными данными:

Текст запроса:

```
USE Authors;
--заполнение бд начальными данными
--заполнение авторов
INSERT INTO _2020_05_authors (id_author, name_author, date_birthday)
    VALUES (1, 'Виктор Петрович', '1993-11-29');
INSERT INTO _2020_05_authors (id_author, name_author, date_birthday)
    VALUES (2, 'Роман Юрьевич', '1995-10-10');
INSERT INTO _2020_05_authors (id_author, name_author, date_birthday)
    VALUES (3, 'Ирина Евгеньевна', '1899-11-19');
INSERT INTO _2020_05_authors (id_author, name_author, date_birthday)
    VALUES (4, 'Любовь Сергеевна', '1996-10-29');
INSERT INTO _2020_05_authors (id_author, name_author, date_birthday)
    VALUES (5, 'Айот Самуель', '1994-10-11');
-- заполнение книг
INSERT INTO _2020_05_books (id_book, name_book, year_book)
    VALUES (1, 'Секреты Oracle SQL', 1995);
INSERT INTO _2020_05_books (id_book, name_book, year_book)
    VALUES (2, 'Системы БД', 1994);
INSERT INTO _2020_05_books (id_book, name_book, year_book)
    VALUES (3, 'MySQL Сборник рецептов', 2005);
INSERT INTO _2020_05_books (id_book, name_book, year_book)
    VALUES (4, 'MangoDB в действии', 1998);
INSERT INTO _2020_05_books (id_book, name_book, year_book)
    VALUES (5, 'PostgresSQL Сборник', 2019);
--заполнение автора и книги
INSERT INTO _2020_05_authors_books (id_authors_books, id_book, id_author)
       VALUES (1, 1, 2);
INSERT INTO _2020_05_authors_books (id_authors_books, id_book, id_author)
       VALUES (2, 2, 1);
INSERT INTO _2020_05_authors_books (id_authors_books, id_book, id_author)
       VALUES (3, 3, 3);
INSERT INTO _2020_05_authors_books (id_authors_books, id_book, id_author)
       VALUES (4, 4, 4);
```

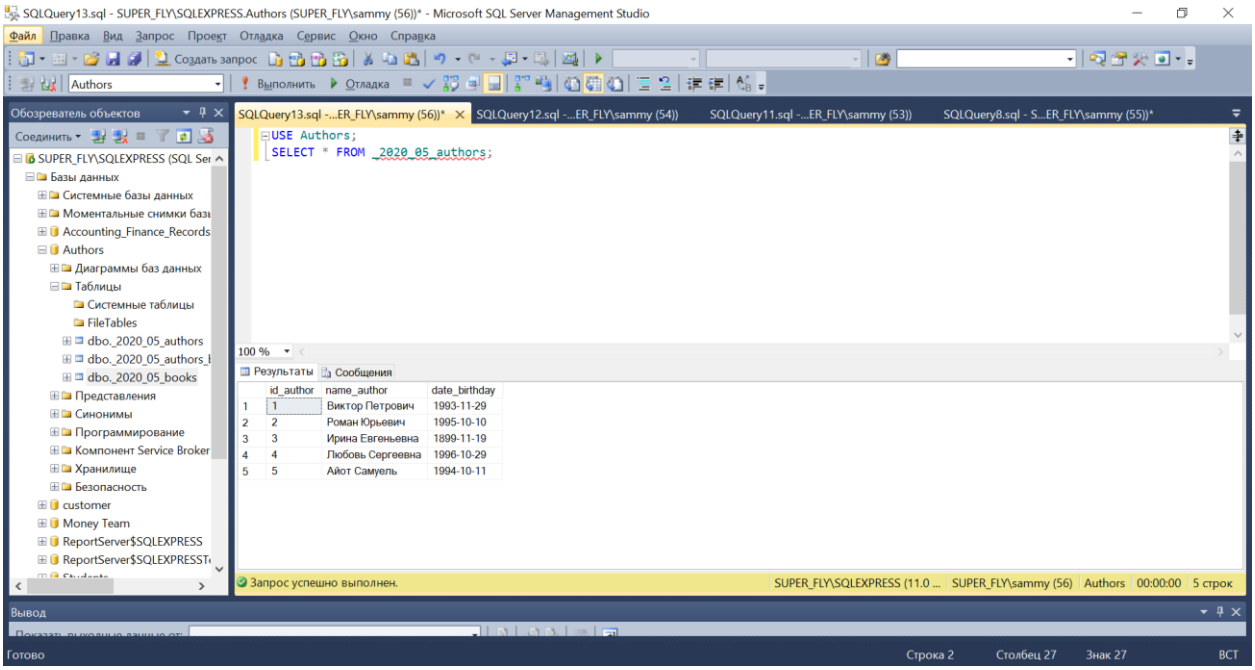
```
INSERT INTO _2020_05_authors_books (id_authors_books, id_book, id_author)
       VALUES (5, 5, 5);
```

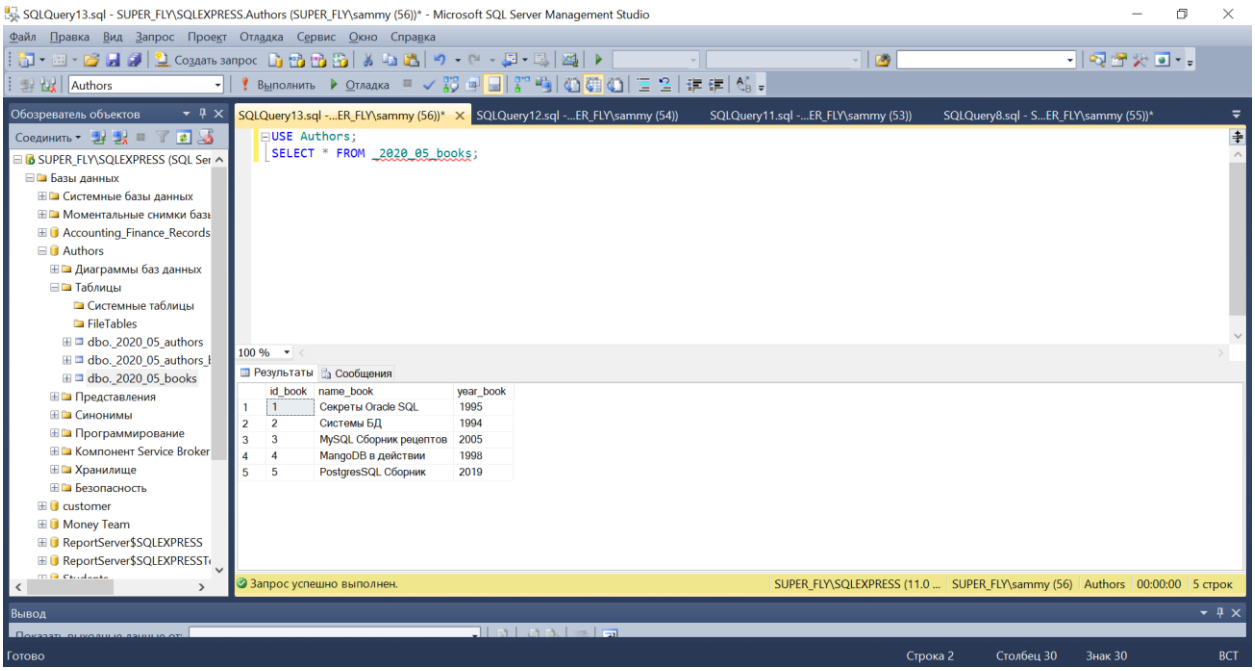
Выберем данные из таблиц:

Текст запроса:

```
--выберем данные из таблиц
SELECT * FROM _2020_05_authors;
SELECT * FROM _2020_05_books;
SELECT * FROM _2020_05_authors_books;
```

Результаты запросов представлены на рисунке 2.





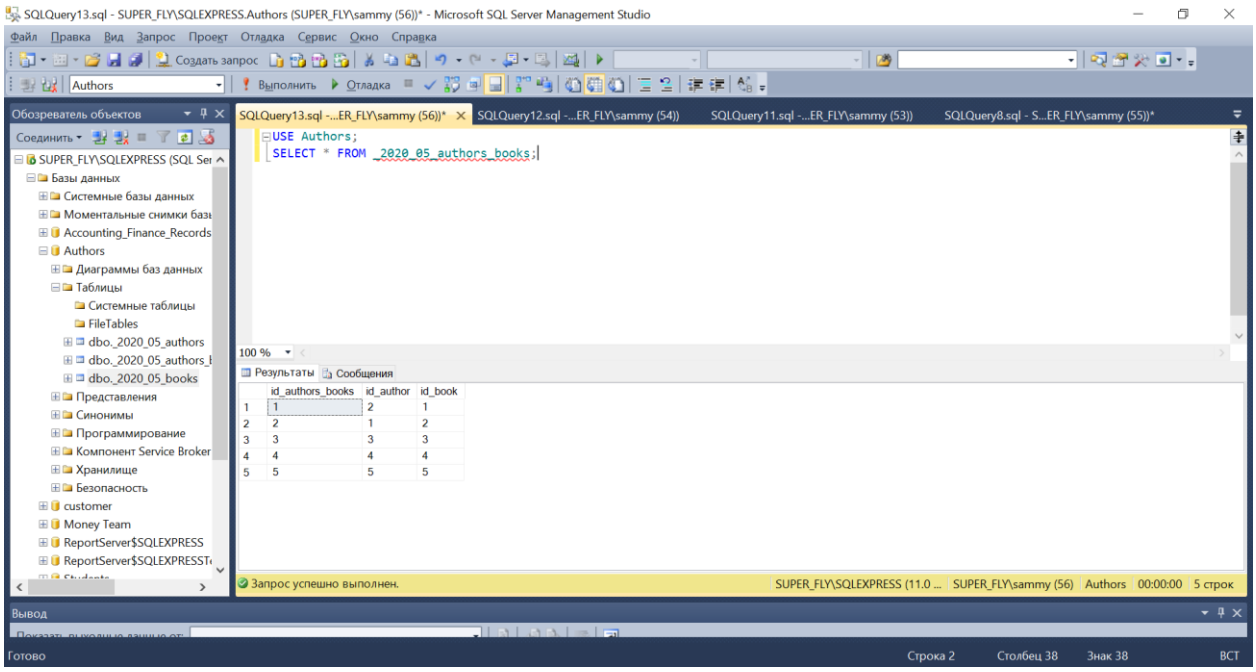


Рисунок 2. – Результаты запросов

Выберем книги и их авторов, которые написаны после 1999 года:

Текст запроса:

```
USE Authors;
SELECT _2020_05_books.name_book, _2020_05_authors.name_author, _2020_05_books.year_book
FROM _2020_05_books
left join _2020_05_authors_books on _2020_05_books.id_book =
_2020_05_authors_books.id_book
left join _2020_05_authors on _2020_05_authors.id_author =
_2020_05_authors_books.id_author
WHERE _2020_05_books.year_book > 1999;
```

Результат выборки представлен на рисунке 3:

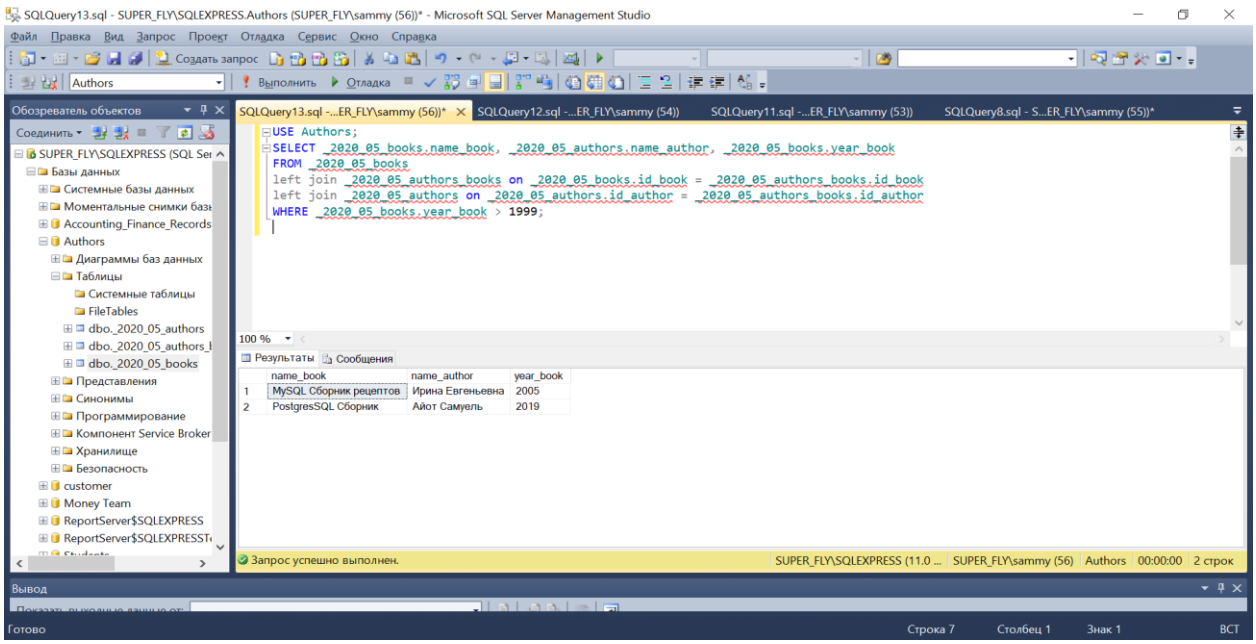


Рисунок 3. – Результат выборки

Вывод:

В работе, было создано представление согласно варианту задания, а также были выполнены примеры предложений, для понимания запросов в SQL.

Хранимые процедуры.

Задание 3: Поиск заданной книги по году выпуска и какому-то слову в названии.

Хранимые процедуры. Описание.

Команда CREATE FUNCTION определяет новую функцию. CREATE OR REPLACE FUNCTION создаёт новую функцию, либо заменяет определение уже существующей. Чтобы определить функцию, необходимо иметь право USAGE для соответствующего языка.

Если указано имя схемы, функция создаётся в заданной схеме, в противном случае — в текущей. Имя новой функции должно отличаться от имён существующих функций с такими же типами аргументов в этой схеме. Однако функции с аргументами разных типов могут иметь одно имя (это называется *перегрузкой*).

Чтобы заменить текущее определение существующей функции, используйте команду CREATE OR REPLACE FUNCTION. Но учтите, что она не позволяет изменить имя или аргументы функции (если попытаться сделать это, на самом деле будет создана новая, независимая функция). Кроме того, CREATE OR REPLACE FUNCTION не позволит изменить тип результата существующей функции. Чтобы сделать это, придётся удалить функцию и создать её заново. (Это означает, что если функция имеет выходные параметры (OUT), то изменить типы параметров OUT можно, только удалив функцию.)

Когда команда CREATE OR REPLACE FUNCTION заменяет существующую функцию, владелец и права доступа к этой функции не меняются. Все другие свойства функции получают значения, задаваемые командой явно или по умолчанию. Чтобы заменить функцию, необходимо быть её владельцем (или быть членом роли-владельца).

Если вы удалите и затем вновь создадите функцию, новая функция станет другой сущностью, отличной от старой; вам потребуется так же

удалить существующие правила, представления, триггеры и т. п., ссылающиеся на старую функцию. Поэтому, чтобы изменить определение функции, сохраняя ссылающиеся на неё объекты, следует использовать CREATE OR REPLACE FUNCTION. Кроме того, многие дополнительные свойства существующей функции можно изменить с помощью ALTER FUNCTION.

Владельцем функции становится создавший её пользователь. Чтобы создать функцию, необходимо иметь право USAGE для типов её аргументов и возвращаемого типа.

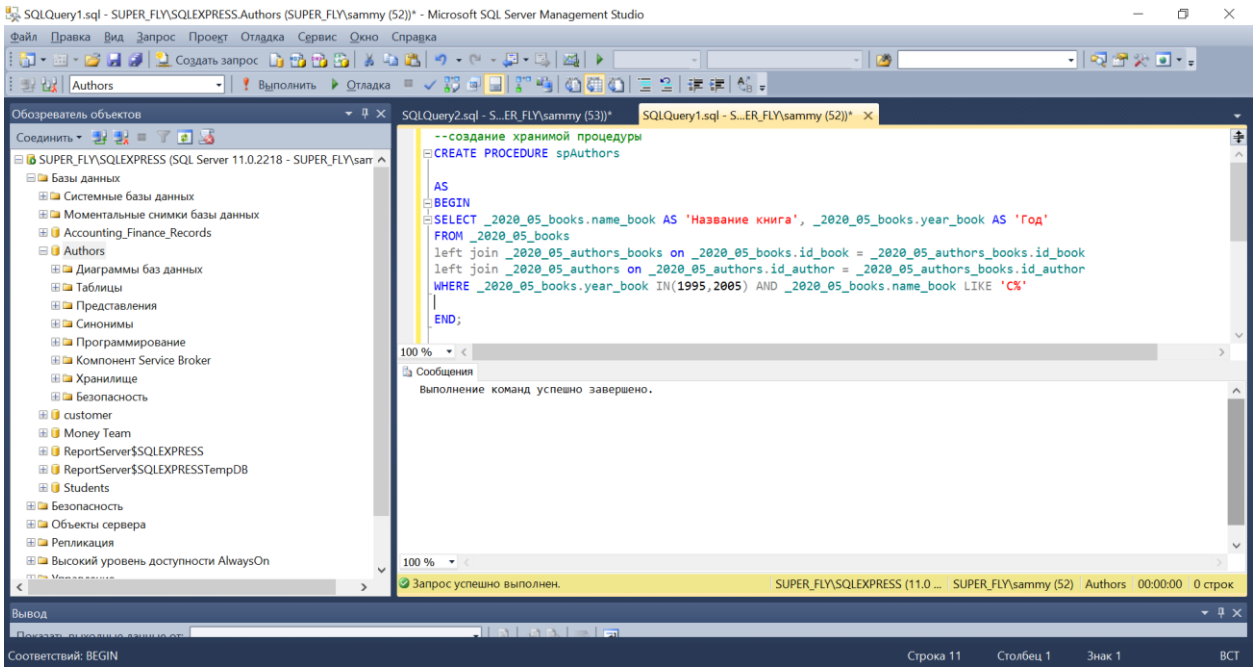
Выполнение:

Запускаем ***Среда SQL Server Management Studio***, и подключаемся к базе данных, созданную из предыдущей лабораторной работы.

Создаем хранимую процедуру по условию задания и вызываем её.

Текст запроса:

```
--создание хранимой процедуры
CREATE PROCEDURE spAuthors

AS
BEGIN
SELECT _2020_05_books.name_book AS 'Название книга', _2020_05_books.year_book AS 'Год'
FROM _2020_05_books
left join _2020_05_authors_books on _2020_05_books.id_book =
_2020_05_authors_books.id_book
left join _2020_05_authors on _2020_05_authors.id_author =
_2020_05_authors_books.id_author
WHERE _2020_05_books.year_book IN(1995,2005) AND _2020_05_books.name_book LIKE 'C%'

END;
```



Результат выполнения представлен на рисунке 1.

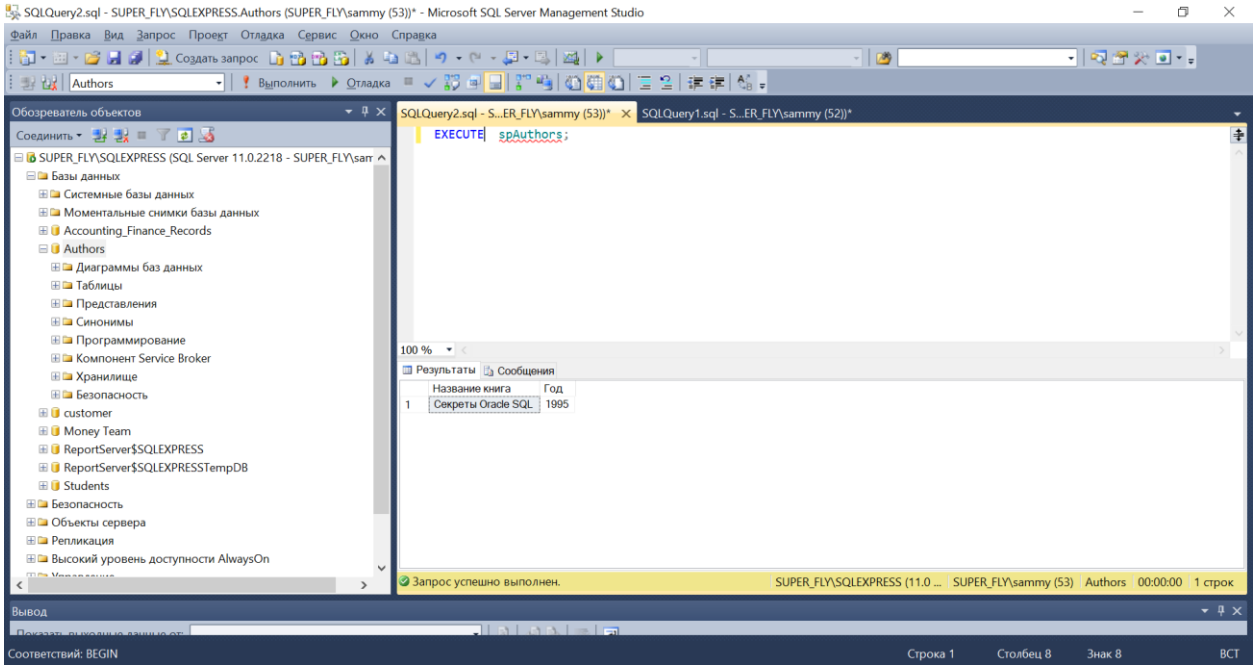


Рисунок 2. – Результат выполнения созданной процедуры

Вывод:

В данной работе, были написаны и отлажены процедуры, согласно прилагаемому заданию для своих таблиц в базе данных. Все процедуры были сохранены на сервер БД.

Гипертекстовые базы данных

Популярность Internet обусловлена возможностью получить доступ ко всем информационным ресурсам с любого компьютера. Технология World Wide Web, в переводе «Всемирная паутина», получила столь широкое распространение из-за простоты своих пользовательских интерфейсов. Принцип «жми на то, что интересно», лежащий в основе гипертекста, интуитивно понятен. В технологиях WWW все ключевые понятия просматриваемого документа: слова, картинки – имеют возможность «раскрыться» новым документом, развивающим это понятие. Для описания этих документов используется специальный язык – язык разметки гипертекстовых документов, или HTML (англ. - HyperText Markup Language). Это не язык программирования, это лишь язык разметки и форматирования документа.

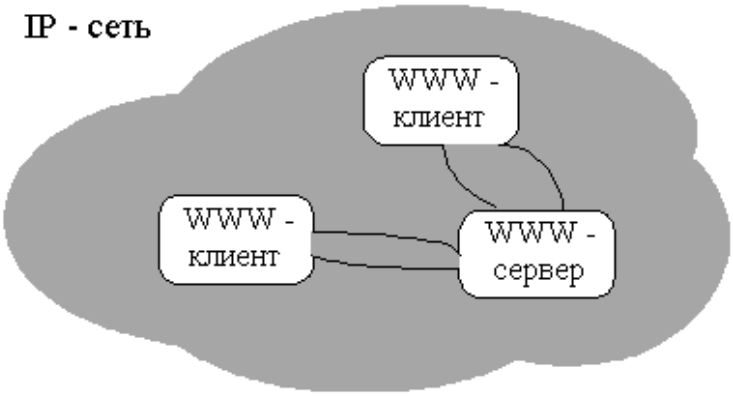
Основные понятия

Использование технологии WWW для обеспечения доступа к каким-либо информационным ресурсам подразумевает существование следующих компонент:

1. IP - сети с поддержкой базового набора услуг по передаче данных с единой политикой нумерации и маршрутизации, работающим сервисом имен DNS.

   Выделенного информационного сервера – ***Web-сервера***, обеспечивающего предоставление гипертекстовых документов через IP-сеть в ответ на запросы ***Web-клиентов***.

Гипертекстовая база данных – совокупность HTML документов, связанных между собой по некоторой схеме с помощью ссылок. Сущность



гипертекстовой БД определяется свойством гипертекстовых документов – любой компонент документа (отдельное слово, текст, рисунок) может быть «раскрыт» другим документом (областью того же документа), дающим более подробную информацию об этом компоненте. Такой способ представления информации и называется гипертекстом.

Достоинства гипертекстовых БД: распределенность – отдельные документы могут находится на разных узлах сети Internet, простота построения и доступа к информации базы данных – не нужно знать языков запросов и моделей баз данных. Но они являются базами данных только в самом широком смысле, как правило это всего лишь совокупность статических файлов, содержащих редко изменяемую слабоструктурированную информацию.

Основу языка HTML составляет стандартный набор тэгов. Существуют парные и непарные тэги. Тэгом является ключевое слово, заключенное в <> (например <HTML>). Парные тэги состоят из открывающего и закрывающего тэга (последний отличается от открывающего символом ”/” перед ключевым словом: например, «</HTML>»). Кроме того тэги могут иметь набор атрибутов, включенных в открывающий тэг (<ключевое_слово атрибут1=значение1 […]>) и определяющих их свойства. Обрабатывая HTML документ, Web-browser (проводник Интернет) – программа, позволяющая

просматривать HTML документы (далее просто проводник) по тэгам определяет как представлять информацию, заключенную между ними, на экран пользователю. Однако проводник не проверяет правильность составления документа, например синтаксис тэгов, наличие закрывающего тэга для парных тэгов. Поэтому иногда можно получить совсем не то, что предполагалось.

Структура документа

Любой HTML документ обрамляется тэгами <HTML> и </HTML>. Структурно HTML документ состоит из двух частей: заголовка (HEAD) и тела (BODY) документа. Шаблон документа:

```
<HTML>
<HEAD>
   Содержание заголовка
</HEAD>
<BODY>
    Содержание тела документа
 </BODY>
</HTML>
```

Заголовок документа может содержать название документа и служебную информацию. Название документа отображается в заголовке проводника и задается внутри парных тэгов <TITLE> … </TITLE> (здесь и далее '…' - информация (текст или другие тэги), которая заключена в этих тэгах).

Тело документа содержит информацию, которую увидит на экране пользователь в виде, заданным тэгами, в которых она заключена. Атрибуты тэга <BODY>: BGCOLOR – цвет фона документа (”FFFFFF” или “предопределенный_цвет”), BACKGROUND - путь до рисунка, используемого в качестве фона (“.\images\1.bmp”).

Содержание документа

Информация в гипертекстовом документе может быть представлена в следующих видах.

Заголовки: <H1> текст заголовка </H1>, …, <H6> ... </H6> Тэги вида <Hi> (где i — цифра от 1 до 6) описывают заголовки шести различных уровней. Заголовок первого уровня — самый крупный, шестого уровня — самый мелкий.

Абзац: <P> ... </P>. Такая пара тэгов описывает абзац. Все, что заключено между <P> и </P>, воспринимается как один абзац.

Тэги <Hi> и <P> могут содержать дополнительный атрибут ALIGN выравнивание по горизонтали, например: <H1 ALIGN=CENTER> по центру</H1>. Значения этого атрибута: LEFT, CENTER, RIGHT, JUSTIFY(выравнивание по левому и правому краю).

Форматированный текст: <PRE> … </PRE>. Текст, заключенный между метками <PRE> и </PRE> (от английского preformatted — предварительно форматированный), выводится проводником на экран как есть — со всеми пробелами, символами табуляции и конца строки.

Для структурирования информации в документах используются списки и таблицы. Различают нумерованные и ненумерованные списки, а также списки определений. Кроме того, списки могут быть вложенными и на вложенность нет никаких ограничений.

Ненумерованные списки: <UL> … </UL>. Внутри этих тэгов могут быть непарные тэги <LH> … - название списка и <LI> … - элемент списка. Название элемента списка можно разбить на несколько абзацев, описанных ранее.

Нумерованные списки: <OL> … </OL>. Внутри этих тэгов могут быть те же тэги, что и для ненумерованных списков.

Список определений: <DL> … </DL>. Внутри этих тэгов могут быть непарные тэги <DT> … - название определения и <DD> … - описание определения.

Для организации вложенных списков после нужного элемента списка вставляются тэги нового списка, например: <LI> текст <OL> … </OL>.

Таблицы: <TABLE> … </TABLE>. Выводит информацию (текст и изображения) в виде таблицы. Возможные атрибуты:

- CAPTION – название таблицы
- BORDER – толщина рамки таблицы в пикселях (0 – без рамки)
- ALIGN – выравнивание таблицы по горизонтали.
- WIDTH – ширина таблицы в пикселях (можно задать в процентах “x%” от ширины экрана)
- BGCOLOR – цвет фона внутри таблицы.
- CELLPADDING – расстояние от границ ячейки таблицы до объекта внутри ячейки.
- CELLSPACING – расстояние между ячейками таблицы.

После открывающего тэга <TABLE> идет список строк таблицы.

Каждая строка это пара тэгов <TR> … </TR>. Для каждой строки можно задать цвет фона (BGCOLOR), горизонтальное выравнивание (ALIGN) и вертикальное выравние (VALIGN, значения “TOP”, “MIDDLE” или “BOTTOM”) внутри ячеек.

В первой строке можно задать имена полей тэгами <TH> … </TH> в виде: <TR> <TH>имя поля1</TH> … <TH> имя поляN</TH> </TR>.

Данные в сроках таблицы задаются тэгами <TD> … </TD>, для всех полей строки. Если поле должно быть пустым, просто ничего не задавайте между этими тэгами. Если вы перечислите меньше значений, чем количество полей, оставшиеся поля также будут пустыми. В качестве значения поля может быть любой текст или рисунок.

Ссылки: это основной элемент, реализующий понятие гипертекста. При помощи ссылок можно переходить к другому документу, области текущего или другого документа. Ссылки задаются тэгами <A> … </A>.

Для перехода к другому документу используется атрибут HREF (Hyper Text Reference). Значением этого атрибута может быть имя документа, к которому необходимо перейти в виде “имя_документа.htm” или полный путь к этому документу, если он находится на другом сервере - “HTTP://…/имя_документа.htm”.

Для перехода к области документа необходимо сначала создать метку или якорь (от англ. слова ancor) с помощью атрибута NAME тэга ссылки, например <A NAME = “метка”>. Для перехода к метке в атрибуте HREF указывается “имя_документа#метка”, а если переход осуществляется внутри документа, то просто “#метка”. Тэг ссылки может иметь как один из этих атрибутов, так и оба атрибута. В последнем случае он является меткой внутри документа и одновременно ссылкой на другой документ (область документа).

Между тэгами ссылки может быть текст, изображение или другие тэги, например:

<A HREF= “имя_документа.htm#метка1” NAME=”метка2”> переход к метке1 </A>. Текст внутри тэгов ссылки на экране будет подчеркнут и выделен другим цветом, а при наведении на него, курсор мыши меняет вид.

Шрифты: Выбор шрифта осуществляется тэгами <FONT FACE="Times New Roman" SIZE="5" COLOR="#0080FF"> … </FONT>. Вся информация, заключенная в эти тэги, будет выводится этим шрифтом. Кроме того, любой текст можно сделать жирным, заключив его в тэги <B> … </B>, курсивом - <I> … </I> или подчеркнутым - <U> … </U>.

Изображения: вставить изображение в документ можно с помощью тэга <IMG SRC="файл_с_изображением" WIDTH="400" HEIGHT="300" ALT="альтернативный_текст">. Атрибут SRC содержит путь до графического файла, WIDTH и HEIGHT задают ширину и высоту изображения, а ALT

содержит текст, который будет выводится вместо изображения, если проводник не поддерживает графику или не найден указанный файл. Если рисунок не соответствует указанным размерам, он будет сжат или расширен.

Разделитель: одиночный тэг <HR> на экране отображается разделительной полосой. Возможные атрибуты ALIGN, WIDTH и SIZE – ширина разделительной полосы.

Перевод строки: Так как в языке HTML пробелы и переводы строки игнорируются везде, кроме форматированного текста, то для принудительного перевода строки в тексте используется непарный тэг
.

Создание документов

Гипертекстовые документы хранятся в виде текстовых файлов с расширением .htm или .html. Создавать документы можно в любом текстовом редакторе, например в Блокноте Windows (Notepad). Для использования созданного документа его необходимо сохранить с соответствующим расширением в каталоге, доступном серверу.

В данном случае это каталог C:\SHTTPS\WWW\. Вам необходимо создать в нем свой каталог с именем №гр№подгр.(например 450501). Все документы сохраняйте в этом каталоге.

Загрузка документа с сервера

В данном курсе лабораторных работ используется свободно распространяемый Small HTTP Server. Эта программа выполняет функции HTTP-сервера. Сервер можно запустить из меню Пуск > Программы > Small HTTP Server. После запуска окно сервера можно свернуть. Загрузка страницы производится из MS Internet Explorer. Запустите IE и в поле адреса введите адрес вашей страницы – HTTP://localhost/ваш_каталог/имя_документа.htm

Задание на лабораторную работу

1. Ознакомитесь со структурой HTML документов и основными тэгами языка HTML.
2. Разработайте структуру гипертекстовой базы данных, содержащей информацию как бы из разработанной вами ранее базы данных. Не пытайтесь получить информацию с сервера – это вы сделаете на следующей работе. Пока от вас требуется продемонстрировать знание основ HTML.
3. По этой структуре создайте гипертекстовую БД, состоящую как минимум из двух HTML документов с использованием имеющихся тэгов для структурирования информации в базе данных.

Создание документов

Гипертекстовые документы хранятся в виде текстовых файлов с расширением .htm или .html. Создавать документы можно в любом текстовом редакторе, например в Блокноте Windows (Notepad). Для использования созданного документа его необходимо сохранить с соответствующим расширением в каталоге, доступном серверу.

Создадим html документ отображающий структуру таблицы authors_book в базе данных. Разметка документа представлена на рисунке 1.

```
<!DOCTYPE html>
<html>
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Authors</title>
    <link rel="stylesheet" href="styles.css" />
</head>
<body>
<table>

    <caption><b>Авторы</b></caption>
    <tr>
        <th>Id_Author</th>
        <th>Name_Author</th>
        <th>Date_Birth</th>
    </tr>
    <tr>
        <td>1</td>
        <td>Виктор Петрович</td>
        <td>1993-11-29</td>
    </tr>
    <tr>
        <td>2</td>
        <td>Роман Юрьевич</td>
        <td>1995-10-20</td>
    </tr>
    <tr>
        <td>3</td>
        <td>Ирина Евгеньевна</td>
        <td>1899-11-19</td>
```

```
    </tr>
    <tr>
        <td>4</td>
        <td>Любовь Сергеевна</td>
        <td>1995-10-29</td>
    </tr>
    <tr>
        <td>5</td>
        <td>Самуель Айот</td>
        <td>1994-10-11</td>
    </tr>
</table>
```

Рисунок 1. Структура документа Авторы

Создадим html документ отображающий структуру таблицы authors в базе данных. Разметка документа представлена на рисунке 2.

```
<table>
    <caption><b>Авторы и Книги</b></caption>
    <tr>
        <th>Id_Authors_Books</th>
        <th>Id_Authors</th>
        <th>Id_Book</th>
    </tr>
    <tr>
        <td>1</td>
        <td>2</td>
        <td>1</td>
    </tr>
    <tr>
        <td>2</td>
        <td>1</td>
        <td>2</td>
    </tr>
    <tr>
        <td>3</td>
        <td>3</td>
        <td>3</td>
    </tr>
    <tr>
        <td>4</td>
        <td>4</td>
        <td>4</td>
    </tr>
    <tr>
        <td>5</thd>
        <td>5</td>
        <td>5</td>
```

```
    </tr>
</table>
</body>
</html>
```

Рисунок 2. Структура документа Авторы и Книги

Создадим html документ отображающий структуру таблицы книги в базе данных. Разметка документа представлена на рисунке 3.

```
<table>
    <caption><b>Книги</b></caption>
    <tr>
        <th>Id_Book</th>
        <th>Name_Book</th>
        <th>Year_Book</th>
    </tr>
    <tr>
        <td>1</td>
        <td>Секреты Oracle SQL</td>
        <td>1995</td>
    </tr>
    <tr>
        <td>2</td>
        <td>Системы БД</td>
        <td>1994</td>
    </tr>
    <tr>
        <td>3</td>
        <td>MySQL Сборник рецептов</td>
        <td>2005</td>
    </tr>
    <tr>
        <td>4</td>
        <td>MangoDB в действии</td>
        <td>1998</td>
    </tr>
    <tr>
        <td>5</td>
        <td>PostgresSQL Сборник</td>
        <td>2019</td>
    </tr>
</table>
```

Рисунок 3. Структура документа Книги

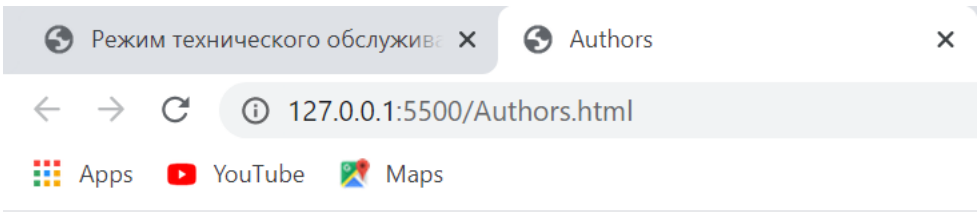
Так же, для корректного отображения документа, необходимо прописать стили для основных тегов документа, таких как: <body />, <table />, <th />, <td />

```
table {
  width: 400px;
}
th {
  text-align: left;
}

table,
th,
td {
  border: 1px solid #000;
  border-collapse: collapse;
}

th,
td {
  padding: 10px;
}
th {
  background-color: rgb(106, 110, 114);
}

td {
  background-color: rgb(66, 182, 12);
}
```

Рисунок 4,Общие стили для всех таблиц

Загрузка документа с сервера

В качестве http сервера, возьмем Live Server. Эта программа выполняет функции HTTP-сервера. Сервер можно запустить из Visual Studio Code. После запуска окно сервера можно свернуть. Загрузка страницы производится из интернет браузера.



Авторы

Id_Author	Name_Author	Date_Birth
1	Виктор Петрович	1993-11-29
2	Роман Юрьевич	1995-10-20
3	Ирина Евгеньевна	1899-11-19
4	Любовь Сергеевна	1995-10-29
5	Самуель Айот	1994-10-11

Рисунок 5. Страница авторов на сервере

Книги

Id_Book	Name_Book	Year_Book
1	Секреты Oracle SQL	1995
2	Системы БД	1994
3	MySQL Сборник рецептов	2005
4	MangoDB в действии	1998
5	PostgresSQL Сборник	2019

Рисунок 6. Страница книг на сервере

Авторы и Книги

Id_Authors_Books	Id_Authors	Id_Book
1	2	1
2	1	2
3	3	3
4	4	4
5	5	5

Рисунок 7. Страница Авторы и книги на сервере

Выводы

В данной работе, были получены навыки работы с html документами, были изучены теги <body>, <table>, <tr> <th >, <td>, а так же применены таблицы стилей к документам. Все документы были запущены на локальном сервере.

ODBC и MFC. CGI-механизм

Вы уже умеете работать с гипертекстом и имеете разработанную и заполненную вами базу данных. В данной работе вам предстоит объединить ваши умения и получить возможность публиковать информацию из базы данных на web.

Для выполнения каких-либо действий на серверной стороне был разработан механизм CGI (Common Gateway Interface). Он основан на том, что при обращении к серверу может быть затребован не только гипертекстовый документ, но и программа (т.н. «CGI-скрипт»). В строке запроса могут быть указаны параметры, необходимые для работы программы, а сама она должна вернуть на стандартный выходной поток (stdout) содержимое гипертекстового документа. Вызов из браузера будет происходить,следующим образом: http://узел/программа.cgi?параметр1=значение1?параметр2=значение2 и так далее. Под управлением систем семейства Windows принято CGI-модулям сообщать расширение .EXE, т.к. в этих системах исполнимые модули распознаются не по атрибуту, как в системах семейства Unix, а по расширению.

Строка параметров, с одной стороны, помещается в переменную окружения QUERY_STRING и может быть получена оттуда, а с другой стороны, параметры эти могут быть получены как обыкновенные параметры командной строки (в Си это параметр argv[] функции main, argc содержит количество параметров вместе с именем программы). В учебном примере используется последний подход.

Не забывайте, что ваша программа должна полностью формировать содержимое гипертекстового документа, от заголовка до последнего закрывающего тэга.

Во введении в курс лабораторных работ (см. язык запросов SQL) была рассмотрена структура сетевой (распределенной СУБД) и упомянут механизм

ODBC. В данной работе предстоит ознакомиться с основами работы с SQL сервером с использованием механизма ODBC.

Механизм ODBC предназначен для организации связи прикладной программы с каким-либо источником данных, в общем случае произвольным. В систему добавляется ODBC-драйвер, который отвечает за соединение с физическим источником данных и за обмен информацией с ним. Прикладная программа общается с источником данных, используя язык SQL. Каждый конкретный драйвер предоставляет возможность в той или иной степени пользоваться возможностями SQL, в зависимости от того, насколько источник это позволяет. Например, при работе с любым SQL сервером (MS SQL Server, Oracle, Interbase, MySQL) возможно пользоваться его языком в полной мере, тогда как при связи с обычной таблицей (Paradox, DBase III+) возможности SQL будут урезаны до операций выборки, добавления и изменения данных, возможно будет предоставлена возможность организации транзакций.

Всякий ODBC-драйвер должен быть зарегистрирован в системе через вкладку **ODBC Data Sources** панели управления Windows. Поскольку системный администратор заблокировал доступ к панели управления системой, вы будете пользоваться готовым ODBC-соединением с именем, совпадающим с номером вашей группы.

В качестве базового языка избран язык Си++ в реализации Microsoft Visual C++.

Классы MFC для работы с источниками данных ODBC

На самом деле, в библиотеке MFC достаточно много мощных средств для работы с источниками данных. Мы рассмотрим лишь два класса, которые будут использоваться для управления источником.

Класс CDatabase

Этот класс представляет собой соединение с базой данных. Соединение устанавливается, затем используется объектом выборки. Данный объект, хотя

и может использоваться для операций с источником данных (например, исполнение SQL-предложений), не возвращает данных, поэтому не может использоваться для их получения. Приведена лишь часть методов, полный перечень находится в документации.

1. CDatabase(); Конструктор. Не принимает параметров и ничего не возвращает.
2. virtual BOOL OpenEx(**LPCTSTR** *lpszConnectString***, DWORD** *dwOptions* **= 0);**

Эта функция устанавливает соединение с источником данных. Первый параметр указывает, с каким источником следует установить соединение, формат следующий:

DSN=<ODBC источник>;UID=<пользователь>;PWD=<пароль>

Указывается наименование источника, зарегистрированного в системе, имя и пароль пользователя на сервере. Если указанной информации недостаточно, будет открыто окно ODBC-соединения. Второй параметр указывает параметры соединения. Вам следует оставить его пустым по умолчанию, т.е. просто опустить при вызове. Функция возвращает ненулевое значение в случае успешного установления соединения.

3. virtual void Close(); Закрывает соединение с источником данных. Всякая работа с ним становится невозможной до повторного открытия.
4. const CString& GetConnect() const; Функция возвращает строку параметров, которая использовалась для установления соединения (см. описание функции OpenEx). Если соединения еще нет, возвращается пустая строка.
5. BOOL IsOpen() const; Возвращает ненулевое значение, если есть активное соединение.
6. BOOL CanTransact() const; Возвращает ненулевое значение, если источник поддерживает транзакции.
7. BOOL BeginTrans( ); Начинает транзакцию на источнике.

8. BOOL CommitTrans( ); Завершает транзакцию на источнике с фиксацией изменений.
9. BOOL Rollback( ); Завершает транзакцию на источнике с откатом изменений.
10. void ExecuteSQL(**LPCSTR** *lpszSQL*); Выполняет SQL-предложение в источнике. Данные не возвращаются, поэтому невозможно ни узнать результат исполнения предложения, ни получить данные.

Класс CRecordset

Данный класс используется для получения данных из источника и засылки данных. Обычно на основе этого класса порождают свои, переопределяя часть методов и добавляя переменные, которые являются полями обмена. Только одно соединение может быть связано с объектом выборки, и только один объект выборки может быть связан с соединением. Этот класс, получая из источника данные, конструирует SQL предложение на основе имеющейся у него информации.

1. CDatabase *m_pDatabase; Указатель на объект соединения типа CDatabase. Если указатель пустой, объект будет создан при открытии выборки.
2. CString m_strFilter; Строка, указывающая область поиска, это аргумент предложения WHERE оператора SELECT. Само слово «WHERE» указывать не следует.
3. CString m_strSort; Строка, указывающая способ сортировки, это аргумент предложения GROUP BY оператора SELECT. Слова «GROUP BY» указывать не следует.
4. virtual BOOL Open(**UINT** *nOpenType* = AFX_DB_USE_DEFAULT_TYPE**, LPCTSTR** *lpszSQL* **= NULL, DWORD** *dwOptions* **= none**); Функция открывает выборку. Вам следует оставить значения по умолчанию, указав в качестве второго параметра имя предложение SELECT для выборки данных.

5. const CString& GetSQL() const; Функция возвращает предложение SQL, использовавшееся для выборки данных.
6. BOOL IsOpen( ) const; Функция возвращает ненулевое значение, если выборка открыта.
7. BOOL IsBOF( ) const; BOOL IsEOF( ) const; Функции возвращают ненулевое значение, если достигнуто начало или конец выборки соответственно.
8. void MoveFirst( ); void MoveLast( ); void MoveNext( ); void MovePrev( ); Функции сдвигают курсор выборки на начало, конец, на следующую и предыдущую запись выборки соответственно. Перед вызовом следует определить положение указателя выборки, чтобы не сдвинуть указатель за начало или конец выборки.
9. virtual BOOL Requery( ); Функция перезапрашивает данные на источнике.
10. long GetRecordCount( ) const; Функция возвращает число записей в текущей выборке.

Пример программы

В нашем примере имеется таблица **[daniil]** в базе данных **_4505**, владелец таблицы – пользователь **_4505**. Таблица имеет следующие поля:

ID типа *integer*, длина 4 байта. В Си это длинное целое;

Name типа *varchar[50]*. В Си это массив символов, строка.

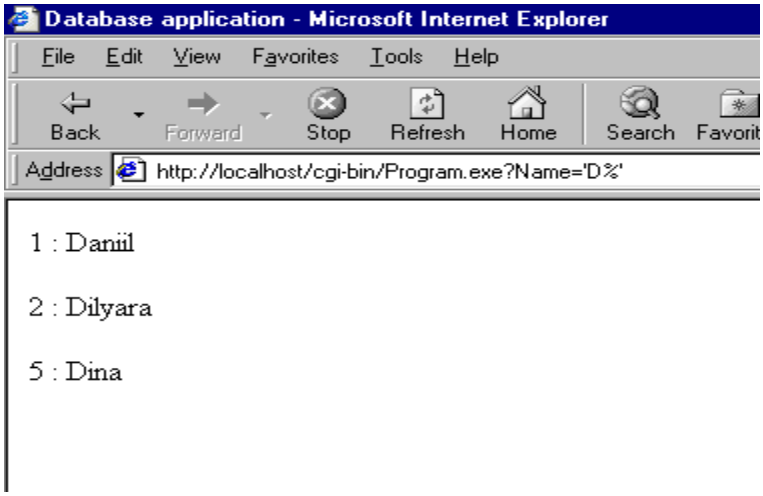
Для работы с данными написана программа, являющаяся консольным приложением системы Windows (работающая в текстовом режиме). Возможности библиотеки MFC используются лишь частично: в частности, не используется порождение нового класса выборки и использование член-переменных класса. Пример состоит из одного модуля main.cpp и представляет собой модуль для CGI-механизма обмена сервера с клиентом. Программа предназначена для вывода на WEB информации из базы данных: записи из таблицы распечатываются построчно с отфильтровыванием

информации по полю Name. Она не организует данные в виде таблиц, не добавляет ссылок и т.п. Тем не менее, это вполне функционально законченная программа, могущая быть использована на web-сервере.

Содержимое таблицы, из которой производится выборка:

ID	Name
1	Daniil
2	Dilyara
3	Jaroslav
4	Marat
5	Dina
6	Elvina
7	Sasha

Результат выполнения программы при обращении к ней с клиентской стороны:



В данном случае при указании маски «D%» выведены все записи с именами, начинающиеся на букву «D».

Текст программы

Это обычное консольное приложение Windows, работающее в текстовом режиме. Не создавайте MDI/SDI программ, диалоговых окон и тому подобных

вещей. CGI-механизм позволяет передавать информацию лишь на уровне http-протокола.

```
#include <iostream.h>
#include <afxdb.h>

int main(int argc, char** argv) {
  // Создаем соединение с базой данных, потребовав не выводить
  // приглашение к соединению и указав все необходимое
  CDatabase db;
  db.OpenEx("DSN=__4505;UID=_4505;PWD=",
CDatabase::noOdbcDialog);
  if (!db.IsOpen()) {
        cout << "Error opening database!\n";
        return 1;
  }

  // Создаем объект выборки, указав предложение для получения
  // данных их таблицы
  CRecordset rs(&db);
  if (!rs.Open(AFX_DB_USE_DEFAULT_TYPE,
                  "SELECT * FROM [_4505].[daniil]",
                  CRecordset::none
  )) {
        cout << "Error opening dataset!\n";
        return 2;
  }

  // Вывод заголовка HTML документа
  cout << "<html>\r\n<head><title>";
```

```
cout << "Database application</title>\n";
cout << "</head>\n";
cout << "<body bgcolor=""#FFFFFF"">\n";

// Проверка параметров. Он должен быть "Name=условие"
if (argc != 2) {
        cout << "ERROR! Bad parameters.\n";
        cout << "</body></html>";
        return 3;
}
if (strncmp(argv[1], "Name=", 5)) {
        cout << "ERROR! Unknown field given.\n";
        cout << "</body></html>";
        return 4;
}
// Проверим, задано ли собственно условие
if (strlen(argv[1]+5) < 1) {
        cout << "<b>Bad parameter, nothing given!\n</b>";
        cout << "</body></html>";
        return 5;
}

// Сформируем фильтр по условию
rs.m_strFilter = "Name LIKE ";
rs.m_strFilter += (argv[1]+5);
rs.Requery();

// Получим данные из выборки
int iID;                // Поле ID
CString sName;    // Поле Name
```

```
    CDBVariant vID;  // Объект для получения данных

    // Если мы не в начале выборки - перейти на начало
    if (!rs.IsBOF()) rs.MoveFirst();

    // Для каждой записи распечатаем ее
    for (; !rs.IsEOF(); rs.MoveNext()) {
        rs.GetFieldValue("ID", vID);
        iID = vID.m_lVal;
        rs.GetFieldValue("Name", sName);
        cout << iID << "  :  " << sName << "<p>";
    }

    // Завершим документ
    cout << "</body></html>";

    // Закроем выборку и базу данных, завершим работу
    rs.Close();
    db.Close();
    return 0;
}
```

В процессе выполнения работы вам предстоит создать свою программу, получающую данные на сервере и отдающую их в виде гипертекстового документа. Проверка будет производиться с использованием web-сервера.

Продумайте, каким образом вы можете получать информацию из вашей базы данных и как ее можно отфильтровывать (или сортировать) по какому-то полю. Разработайте механизм взаимодействия клиентской и серверной частей. Напишите и отладьте CGI-скрипт, поместите его на сервер и обратитесь к нему с различными параметрами. Помните, что необходимо следить за их корректностью и извещать пользователя, если параметры заданы некорректно:

сервер может быть удален от пользователя на тысячи километров и последний видит лишь то, что передал сервер в момент взаимодействия.

Задание 5

Создать программу, получающую данные на сервере и отдающую их в виде гипертекстового документа. Проверку произвести с использованием web-сервера.

Подготовка документов

Изменим html документы, созданные в работе №4. Для заполнения их данными, будем использовать шаблонизатор ejs, Дополним их метками, в которые будут вставляться данные, полученные из базы данных. Разметка документа представлена на рисунке 1.

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="utf-8">
    <link rel="stylesheet" href="./styles.css"></style>
    <title><%= title %></title>
  </head>
  <body>
    <main>
      <h1><%= header %></h1>
      <table>
        <tr>
          <th>id</th><th>id автора</th><th>id книги</th>
        </tr>
          <%- tableBody %>
      </table>
    <main>
  </body>
</html>
```

Рисунок 1. Структура документа Авторы и книги

Изменим html документ отображающий структуру таблицы authors в базе данных. Разметка документа представлена на рисунке 2.

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="utf-8">
    <link rel="stylesheet" href="./styles.css"></style>
    <title><%= title %></title>
  </head>
  <body>
    <main>
      <h1><%= header %></h1>
      <table>
        <tr>
          <th>id</th><th>Имя</th><th>Дата рождения</th>
        </tr>
        <%- tableBody %>
      </table>
    <main>
  </body>
</html>
```

Рисунок 2. Структура документа Авторы

Изменим html документ, отображающий структуру таблицы books в базе данных. Разметка документа представлена на рисунке 3.

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="utf-8">
    <title><%= title %></title>
    <link rel="stylesheet" href="./styles.css"></style>
  </head>
  <body>
    <main>
      <h1><%= header %></h1>
      <table>
        <tr>
          <th>id</th><th>Название</th><th>Дата издания</th>
        </tr>
        <%- tableBody %>
      </table>
    <main>
  </body>
</html>

```

Рисунок 3. Структура документа Книги

Разработка сервера

Серверное приложение, разработано на node.js с использованием фреймворка express

```
var express = require('express');
var ejs = require('ejs');
var fs = require('fs');

var app = express();
const promise = require('bluebird');

var options = {
    promiseLib: promise
};

var pgp = require('pg-promise')(options);
var cn = {
    host: 'localhost',
    port: 5432,
    database: 'staging_db',
    user: 'postgres',
    password: '123456'
};
var db = pgp(cn);

app.set('view engine', 'html');
app.use(express.static(__dirname + '/public'));

```

Рисунок 4. Инициализация сервера

```
app.get('/books', function (req, res) {
  var where = req.query.name || req.query.year ? 'where ' : ''
  var name = req.query.name ? "name_book LIKE '%" + req.query.name + "%' " : ''
  var and = req.query.year ? 'and' : ''
  var year = req.query.year ? 'year_book='+req.query.year : '';
  var query = 'select * from _2018_06_books ' + where + name + and + ' ' + year;
  db.any(query)
    .then(function (data) {
      var table = ''
      data.map(function(col) {
        table += `
          <tr>
            <td>${col.id_book}</td>
            <td>${col.name_book}</td>
            <td>${col.year_book}</td>
          </tr>
        `;
      });
      var compiled = ejs.compile(fs.readFileSync(__dirname + '/public/books.html', 'utf8'));
      var html = compiled({ title : 'Books', header : 'Таблица книги', tableBody: table  });
      res.send(html);
    })
    .catch(function (error) {
        console.log("ERROR:", error);
    });
});
```

Рисунок 5. Запрос на таблицу книг

```
app.get('/authors', function (req, res) {
  console.log(req.query);
  var where = req.query.name ? 'where ' : ''
  var name = req.query.name ? "name_author LIKE '%" + req.query.name + "%' " : '';
  // var and = req.query.name ? 'and' : '';
  // var birthday = req.query.birthday ? "date_birthday" + req.query.birthday.toString() : '';
  var query = 'select * from _2018_06_authors ' + where + name;
    console.log(query);
  db.any(query)
      .then(function (data) {
          var table = ''
          data.map(function(col) {
            table += `
              <tr>
                <td>${col.id_author}</td>
                <td>${col.name_author}</td>
                <td>${col.date_birthday}</td>
              </tr>
            `;
          });
          var compiled = ejs.compile(fs.readFileSync(__dirname + '/public/authors.html', 'utf8'));
          var html = compiled({ title : 'Authors', header : 'Таблица авторов', tableBody: table });
          res.send(html);
      })
      .catch(function (error) {
          console.log("ERROR:", error);
      });
  // res.sendfile(__dirname + '/books.html');
});
```

Рисунок 6. Запрос на таблицу авторов

```
app.get('/authors_book', function (req, res) {
  db.any("select * from _2018_06_authors_books")
      .then(function (data) {
          var table = ''
          data.map(function(col) {
            table += `
              <tr>
                <td>${col.id_authors_books}</td>
                <td>${col.id_author}</td>
                <td>${col.id_book}</td>
              </tr>
            `;
          });
          var compiled = ejs.compile(fs.readFileSync(__dirname + '/public/authors_book.html', 'utf8'));
          var html = compiled({ title : 'Authors Books', header : 'Таблица авторов и книг', tableBody: table });
          res.send(html);
      })
      .catch(function (error) {
          console.log("ERROR:", error);
      });
});

app.listen(3000, function () {
  console.log('Example app listening on port 3000!');
})
```

Рисунок 7. Запрос на таблицу авторов и книг

Таблица книги

Id	Название	Дата издания
1	Секреты Oracle SQL	1995

Рисунок 8. Отображение данных из БД на html документ

Таблица книги

Id	Название	Дата издания
3	MySQL Сборник рецептов	2005

Рисунок 9. Отображение данных из БД на html документ

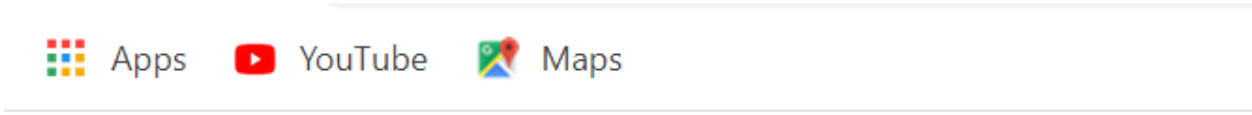


Таблица авторов

Id	Имя	Дата Рождения
1	Виктор Петрович	Ср Ноя 29 1993 00:00:00 GMT+0300(MSK)
2	Роман Юрьевич	Вт Окт 10 1995 00:00:00 GMT+0300(MSK)
3	Ирина Евгеньевна	Вс Ноя 11 1899 00:00:00 GMT+0300(MSK)
4	Любовь Сергеевна	Сб Янв 29 1996 00:00:00 GMT+0300(MSK)
5	Самуель Айот	Пт Окт 11 1994 00:00:00 GMT+0300(MSK)

Рисунок 10. Отображение данных из БД на html документ

Выводы

В данной работе, была создана программа, получающая данные на сервере и отдающую их в виде гипертекстового документа. Работа программы проверена с использованием web-сервера.