Практическая работа 46
Организация выборки данных с помощью SQL запроса. Создание приложения базы данных

Цель работы: Получить практический опыт создания приложения базы данных.

Перечень оборудования и программного обеспечения

Персональный компьютер
Microsoft Office (Word, Visio, Access)

Краткие теоретические сведения

Запросы являются мощным средством обработки данных, хранимых в таблицах Access. С помощью запросов можно просматривать, анализировать и изменять данные из нескольких таблиц. Они также используются в качестве источника данных для форм и отчетов. Запросы позволяют вычислять итоговые значения и выводить их в компактном формате, подобном формату электронной таблицы, а также выполнять, вычисления над группами записей.

Запросы можно создавать самостоятельно и с помощью мастеров. Мастера запросов автоматически выполняют основные действия в зависимости от ответов пользователя на поставленные вопросы. Самостоятельно разработать запросы можно в режиме конструктора.

В Access можно создавать следующие типы запросов:

запрос на выборку;

запрос на изменение (запрос на удаление, обновление и добавление записей на создание таблицы);

запросы SQL (запросы на объединение, запросы к серверу, управляющие запросы, подчиненные запросы)

SQL — это язык, предназначенный для работы с наборами фактов и отношениями между ними. В программах управления реляционными базами данных, таких как Microsoft Office Access, язык SQL используется для работы с данными. Как и многие языки программирования, SQL является международным стандартом, признанным такими комитетами по стандартизации, как ISO и ANSI.

Возможности SQL запросов:

1 Создание, изменение и удаление таблиц БД;
2 Выборка информации из таблиц;
3 Ввод записей в таблицы БД;
4 Редактирование записей в таблицах;
5 Удаление записей из БД.

При использовании SQL необходимо применять правильный синтаксис. Синтаксис — это набор правил, позволяющих правильно сочетать элементы языка.

Инструкция SQL состоит из нескольких частей, называемых предложениями. Каждое предложение в инструкции SQL имеет свое назначение. Некоторые предложения являются обязательными. В приведенной ниже таблице указаны предложения SQL, используемые чаще всего.

Предложение SQL	**Описание**	**Обязательное**
SELECT	Определяет поля, которые содержат нужные данные.	Да
FROM	Определяет таблицы, которые содержат поля, указанные в предложении SELECT.	Да
WHERE	Определяет условия отбора полей, которым должны соответствовать все записи, включаемые в результаты.	Нет
ORDER BY	Определяет порядок сортировки результатов.	Нет
GROUP BY	Осуществляет группировку по какому либо полю, что иногда является необходимым. И уже для этой группы производит заданное действие.	Только при наличии таких полей
HAVING	Используется как дополнение к предыдущему. В инструкции SQL, которая содержит статистические функции, определяет условия, применяемые к полям, для которых в предложении SELECT вычисляется сводное значение.	Нет

Оператор SQL состоит из зарезервированных слов, а также со слов, которые определяются пользователем.

Зарезервированные слова являются постоянной частью речи SQL и имеют фиксированное значение. Их следует записывать в точности так, как это установлено, нельзя разбивать на части для переноса из одной строки на другую. Слова, которые определяются пользователем, задаются им самим (соответственно с синтаксическими правилами) и являют собой идентификаторы или имена разных объектов базы данных. Слова в операторе размещаются также в соответствии с установленными синтаксическими правилами.

Для записи выражений SQL будем следовать следующим соглашениям:

– прописные буквы используются для записи зарезервированных слов;
– малые - для записи параметров;

– вертикальная черта Необходимость выбора одного из нескольких приведенных значений;
– фигурные скобки содержат обязательный элемент выражения;
– квадратные скобки содержат необязательный элемент выражения;
– многоточие ".". значит повторение ранее отмеченной конструкции нуль или больше раз
– строчные константы записываются в одинарных кавычках.
– несущественные операнды и элементы синтаксиса (например, принятое во многих системах программирование правило ставить ";" в конце оператора) будем опускать.

Стандарт SQL использует терминологию "таблица", "столбец", "строка".

Синтаксис запроса на выборку:

SELECT поле1, поле2, поле3 …
FROM таблица (если нужны все поля то ставим *)
WHERE условие1 AND/OR условие 2 ….
В условиях используем операции сравнения =, <>, >, <, >=, <=, LIKE, BETWEEN, IN

Например, простая инструкция SQL, извлекающая список фамилий контактов с именем Mary, может выглядеть следующим образом:

```
SELECT Last_Name
FROM Contacts
WHERE First_Name = 'Mary';
```

В запросах по нескольким таблицам используем явное определение полей. Начнем с самого простого способа, в котором будем явно указывать поля и условия, при котором их нужно выводить.

Вот пример: **Вывести попарно продавцов и покупателей из одного города.** Поскольку, нужно вывести попарно то придется перебрать все комбинации — SQL сделает это:

```
SELECT salespeople.sname, customers.cname, customers.city
FROM salespeople, customers
WHERE salespeople.city = customers.city
```

Сортировка

ORDER BY поле ASC/DESC

Группировка

GROUP BY поле

Условие

HAVING условие

В условии можно использовать агрегатные функции. Эти функции выполняются с помощью ключевых слов, которые включаются в запрос

SELECT. Чтобы было понятно, вот некоторые возможности агрегатных функций в SQL:

- Суммировать выбранные значения
- Находить среднее арифметическое значений
- Находить минимальное и максимальное из значений

Наиболее часто используемые функции:

Функция SUM

Эта функция позволяет просуммировать значения какого либо поля при запросе SELECT. Достаточно полезная функция, синтаксис которой довольно прост, как и всех других агрегатных функций в SQL. Для понимания сразу начнем с примера:

Получить сумму всех заказов из таблицы Orders, которые были совершены в 2016 году.

Можно было бы просто вывести сумму заказов, но мне кажется, что это совсем просто. Напомним структуру нашей таблицы:

onum	amt	odate	cnum	snum
1001	128	2016-01-01	9	4
1002	1800	2016-04-10	10	7
1003	348	2017-04-08	2	1
1004	500	2016-06-07	3	3
1005	499	2017-12-04	5	4
1006	320	2016-03-03	5	4
1007	80	2017-09-02	7	1
1008	780	2016-03-07	1	3
1009	560	2017-10-07	3	7
1010	900	2016-01-08	6	8

Следующий код осуществит нужную выборку:

```
SELECT SUM(amt)
FROM Orders
WHERE odate BETWEEN '2016-01-01' and '2016-12-31';
```

В результате получим:

SUM(amt)
4428

В данном запросе мы использовали **функцию SUM**, после которой в скобках нужно указать поле для суммирования. Затем мы указали условие в WHERE, которое отобрало строчки только с 2016 годом. На самом деле это условие можно записать по другому, но сейчас важнее агрегатная функция суммирования в SQL.

Функция AVG

Следующая функция осуществляет подсчет среднего арифметического поля данных, которое мы укажем в качестве параметра. Синтаксис такой функции идентичен функции суммирования. Поэтому сразу перейдем к простейшей задаче:

Вывести среднюю стоимость заказа из таблицы Orders.

И сразу запрос:

SELECT AVG(amt)
FROM Orders;

В результате получим:

AVG(amt)
591.5

В целом, все похоже на предыдущую функцию. И синтаксис достаточно прост. В этом и состоит особенность языка SQL — быть понятным для человека.

Функции MIN и MAX

Еще 2 функции, которые близки по своему действию. Они находят минимальное или максимальное значение соответственно того параметра, который будет передан в скобках. Синтаксис повторяется и поэтому следующий пример:

Вывести максимальное и минимальное значения цены заказа, для тех заказов в которых цена менее 1000.

Получается такой запрос,

SELECT MAX(amt), **MIN**(amt)
FROM Orders
WHERE amt < 1000;

который выведет:

MAX(amt)	MIN(amt)
900	80

Также стоит сказать, что в отличие от предыдущих функций, эти 2 могут работать с символьными параметрами, то есть можно написать запрос типа **MIN(odate)** (в данном случае дата у нас символьная), и тогда нам вернется 2016-01-01.

Дело в том, что в этих функциях есть механизм преобразования символов в ASCII код, который потом они и сравнивают.

Еще одним важным моментом является то, что мы можем производить некоторые простые математические операции в запросе SELECT, например, такой запрос:

SELECT (**MAX**(amt) - **MIN**(amt)) **AS** 'Разница'
FROM Orders;

Вернет такой ответ:

Разница
1720

Функция COUNT

Эта функция необходима для того, чтобы подсчитать количество выбранных значений или строк. Существует два основных варианта ее использования:

– С ключевым словом DISTINCT, для того, чтобы подсчитать количество не повторяющихся значений

– С использованием «*», для того, чтобы подсчитать количество всех выбранных значений

Теперь разберем пример использования COUNT в SQL:

Подсчитать количество сделанных заказов и количество продавцов в таблице Orders.

SELECT COUNT(*), **COUNT**(**DISTINCT** snum)
FROM Orders;

Получаем:

COUNT(*)	COUNT(snum)
10	5

Очевидно, что количество заказов — 10, но если вдруг у вас имеется большая таблица, то такая функция будет очень удобной. Что касается уникальных продавцов, то здесь необходимо использовать DISTINCT, потому что один продавец может обслужить несколько заказов.

Оператор GROUP BY

Теперь рассмотрим 2 важных оператора, которые помогают расширить функционал наших запросов в SQL. Первым из них является оператор GROUP BY, который осуществляет группировку по какому либо полю, что иногда является необходимым. И уже для этой группы производит заданное действие. Например:

Вывести сумму всех заказов для каждого продавца по отдельности.

То есть теперь нам нужно для каждого продавца в таблице Orders выделить поля с ценой заказа и просуммировать. Все это сделает оператор GROUP BY в SQL достаточно легко:

SELECT snum, **SUM**(amt) **AS** 'Сумма всех заказов'
FROM Orders
GROUP BY snum;

И в итоге получим:

snum	Сумма всех заказов
1	428
3	1280
4	947
7	2360
8	900

Как видно, SQL выделил группу для каждого продавца и посчитал сумму всех их заказов.

Оператор HAVING

Этот оператор используется как дополнение к предыдущему. Он необходим для того, чтобы ставить условия для выборки данных при группировке. Если условие выполняется то выделяется группа, если нет — то ничего не произойдет. Рассмотрим следующий код:

SELECT snum, **SUM**(amt) **AS** 'Сумма всех заказов'
FROM Orders
GROUP BY snum

HAVING MAX(amt) > 1000;

Который создаст группу для продавца и посчитает сумму заказов этой группы, только в том случае, если максимальная сумма заказа больше 1000. Очевидно, что такой продавец только один, для него выделится группа и посчитается сумма всех заказов:

snum	**Сумма всех заказов**
7	2360

Казалось бы, почему тут не использовать условие WHERE, но SQL так построен, что в таком случае выдаст ошибку, и именно поэтому в SQL есть оператор HAVING.

Примеры на агрегатные функции в SQL

1. Напишите запрос, который сосчитал бы все суммы заказов, выполненных 1 января 2016 года.

SELECT SUM(amt)
FROM Orders
WHERE odate = '2016-01-01';

2. Напишите запрос, который сосчитал бы число различных, отличных от NULL значений поля city в таблице заказчиков.

SELECT COUNT(**DISTINCT** city)
FROM customers;

3. Напишите запрос, который выбрал бы наименьшую сумму для каждого заказчика.

SELECT cnum, **MIN**(amt)
FROM orders
GROUP BY cnum;

4. Напишите запрос, который бы выбирал заказчиков, чьи имена начинаются с буквы Г.

SELECT cname
FROM customers
WHERE cname **LIKE** 'Г%' ;

5. Напишите запрос, который выбрал бы высший рейтинг в каждом городе.

SELECT city, **MAX**(rating)
FROM customers
GROUP BY city;

Разработка приложений баз данных для MS Access

Модули, в отличие от макросов, являются более тонким и мощным средством создания программных расширений в среде Access, максимально приближающимся по своим функциональным возможностям к таким профессиональным инструментам, как Delphi, Visual Basic или Power Builder. Одновременно применение модулей требует от пользователя навыков и квалификации программиста, а также знания основных принципов объектно-ориентированного программирования.

Для программирования в Access используется процедурный язык Visual Basic для приложений (VBA- Visual Basic for Applications) с добавлением объектных расширений и элементов SQL. Сам процесс создания программных расширений в среде Access предполагает активное использование технологии объектно-ориентированного программирования (ООП). В основе ООП лежит идея "упакованной функциональности", в соответствии с которой программа строится из фундаментальных сущностей, называемых объектами. Каждый из объектов характеризуется набором свойств (англ, -property) и операций, которые он может выполнять (англ,-method). Реализация взаимодействий между объектами ложится на исполняющую среду того средства разработки, на котором пишется программа, и поэтому работа программиста в рамках технологии ООП сводится к созданию объектов, описанию их свойств и реакций на те иди иные внешние события.

Фундаментальным понятием ООП является класс. Класс - это шаблон, на основе которого может быть создан конкретный программный объект. Созданный объект в таком случае становится экземпляром класса. К основополагающим принципам ООП относятся:

" инкапсуляция - объединение свойств и действий, присущих объекту, в едином пакете и сокрытие подробностей их реализации от окружающего мира. Это означает, что пользовательский доступ к объекту допускается только через его свойства и методы;

" наследование - предусматривает создание новых классов на базе существующих, что дает возможность классу-потомку иметь (наследовать) все свойства класса-родителя;

" полиморфизм - (от греч. "многоликость") означает, что порожденные объекты обладают информацией о том, какие методы они должны использовать в зависимости от того, где они находятся в цепочке наследования;

" модульность - объекты заключают в себе полное определение их характеристик, никакие определения методов и свойств объекта не должны располагаться вне его, что делает возможным свободное копирование и внедрение одного объекта в другие.

Многие программные объекты в Access совпадают с физическими объектами базы данных, такими как таблицы, формы, отчеты. Для названия составных объектов, которые включают в себя совокупность более простых объектов, используется термин семейство. Например, объект отчет входит в семейство отчеты. Помимо "видимых" объектов существует и большое количество "скрытых" объектов, управлять которыми можно только из программных приложений.

В Access существуют два типа модулей: стандартные и модули класса. Стандартные модули содержат процедуры и функции, которые могут быть вызваны из любого окна базы данных. Как правило, такие модули содержат программный код универсального характера, предназначенный для применения в различных местах текущего приложения или даже в различных

приложениях.
Модули класса используются, для создания новых классов объектов. При создании конкретного объекта, являющегося экземпляром такого класса, любые процедуры, определенные в модуле, становятся свойствами и методами этого объекта.

Модули форм и модули отчетов являются модулями класса, связанными с определенной формой или отчетом. Заметим, что в ранних версиях Access они являлись единственно возможным инструментом объектно-ориентированного программирования. Эти модули содержат процедуры обработки событий, запускаемых в ответ на их возникновение в форме или отчете. Процедуры обработки событий используются для управления поведением формы или отчета и их откликом на события, например такие, как нажатие кнопки.

Важнейшей областью применения объектно-ориентированного программирования в Access является программирование доступа к данным. Для решения данной задачи фирмой Microsoft был разработан специальный интерфейс - DAO (Data Access Objects).

DAO - это набор объектных классов, которые моделируют структуру реляционной базы данных. Они обеспечивают свойства и методы, которые позволяют выполнять такие операции, как создание базы данных, определение таблиц и индексов, задание связей между таблицами, формирование запросов и отчетов и т. п. Существенным достоинством объектной модели DAO является ее универсальный характер: она доступна для большинства средств разработки программного обеспечения, поддерживаемых Microsoft, в частности, для Visual Basic. Классы объектов доступа к данным организованы по иерархической схеме. На ее вершине находится объект DbEngine, представляющий собой ядро базы данных.

Далее следуют объекты, отвечающие за управление сеансами доступа пользователя к данным, - Workspace (от англ, "рабочая область"). Каждая рабочая область включает один или несколько объектов класса база данных - Database, а они, в свою очередь, содержат семейства объектов таблиц (TableDef), запросов (QueryDef), наборов записей (RecordSet) и т. д.

Наиболее современной технологией разработки приложений, управляющих данными, является построение многоуровневых систем:

1. **Уровень "Хранилище данных"** (*Data Store*) - место, где находятся сами данные. Это может быть реляционная база данных, XML-файл, текстовый файл или какая-то другая структурированная система хранения .

2. **Уровень "Доступ к данным"** (Data Access Layer - DAL) - код, который необходим для извлечения и манипулирования необработанными данными, находящимися в хранилище.

3. **Уровень "Бизнес-логика"** (*Business Logic* Layer - BLL) - код, который берет извлеченные данные и предоставляет их клиенту в более понятном виде, а также обеспечивает безопасность и согласованность действий клиента и данных.

4. **Уровень "Представление"** ("Пользовательский интерфейс") (Presentation/User Interface Layer - UI) - код, который определяет, что именно должен видеть пользователь на экране, включая форматирование данных и навигационное меню системы.

Доступ к данным обеспечивает обобщенный *поставщик данных*, который включает в себя следующие классы:

- **DbConnection** - используется для установки соединения с источником данных
- **DbCommand** - используется для выполнения SQL-команд и хранимых процедур
- **DbDataReader** - модуль чтения, который предоставляет быстрый последовательный доступ к данным только для чтения. Он сам автоматически не управляет открытием и закрытием соединения и это нужно делать вручную, открывая как можно позже и закрывая как можно раньше
- **DbDataAdapter** - выполняет две задачи:

1. Наполнение набора данных **DataSet** (автономная коллекция таблиц и отношений) информацией, извлеченной из источника данных
2. Применение изменений данных, выполненных пользователем в **DataSet**, к источнику данных

Этот обобщенный поставщик напрямую не применяется, а наследуется специализированными поставщиками, имена которых имеют префиксы Odbc, OleDb, Sql, Oracle. Мы будем использовать набор классов OleDb для файловой *БД* Northwind.mdb. *Нужно четко понимать, что поставщики ADO.NET отображают реляционную модель данных хранилища в объектно-ориентированную модель приложения.* Это значит, что данные таблицы или *множества* таблиц после их считывания приложением представляются наборами взаимосвязанных объектов соответствующих типов и дальнейшая работа проводится уже с этими объектами.

Объект **DbCommand** может выполнять три типа команд, определяемых перечислением CommandType:

1. **Text** - команда будет выполнять прямой SQL-оператор, который мы укажем в свойстве **DbCommand.CommandText**
2. **StoredProcedure** - команда будет выполнять созданную нами ранее процедуру, хранимую в источнике данных. Свойство **DbCommand.CommandText** представляет имя хранимой процедуры
3. **TableDirect** - команда извлечет все записи указанной в свойстве **DbCommand.CommandText** таблицы подключенной БД (не работает с поставщиком SQL Server)

Для выполнения созданного и настроенного объекта **DbCommand** он имеет три команды:

1. **ExecuteReader()** - выполняет запрос **SELECT** и возвращает объект **DbDataReader**, который является оболочкой однонаправленного курсора, доступного только для чтения

2. **ExecuteScalar()** - выполняет запрос SELECT и возвращает значение первого поля первой строки из набора строк, сгенерированного командой. Этот метод обычно применяется при исполнении агрегатной команды **SELECT**, использующей функции наподобие **COUNT()** или **SUM()** для вычисления единственного значения

3. **ExecuteNonQuery()** - для выполнения незапросной команды, которая не требует выборки данных с помощью SQL-оператора **SELECT**. Применяется для исполнения SQL-команд вставки, удаления или обновления записей. Возвращаемое значение означает количество строк, обработанное командой

Модуль чтения **DbDataReader** способен загрузить в себя данные из нескольких таблиц одной *БД*, т.е. получить несколько наборов результатов. *Доступ* к считанным наборам результатов можно получить с помощью методов **DbDataReader**. Вот некоторые из них:

<table>
<tr><th colspan="2">Некоторые методы DbDataReader</th></tr>
<tr><th>Метод</th><th>Описание</th></tr>
<tr><td>bool Read()</td><td>Перемещает курсор строки результирующего набора на следующую строку. Этот метод должен быть вызван, также, перед чтением первой строки данных, т.к. после наполненияDbDataReaderданными курсор устанавливается перед первой строкой. Метод возвращает true, если есть куда переводить, при применении к последней строке он вернет false</td></tr>
<tr><td>object GetValue(intordinal)</td><td>Возвращает значение поля текущей строки с указанным индексом столбца. Тип возвращенного значения заменяется типом .NET, наиболее подходящим типу столбца в хранилище</td></tr>
<tr><td>int GetValues(object[ ] values)</td><td>Заполняет массив values полями текущей записи. Количество полей определяется размером массива, который можно определить по свойству FieldCount</td></tr>
<tr><td>int GetInt32(intordinal)</td><td rowspan="4">Эти методы возвращают значение поля с указанным индексом в текущей строке (нумерация полей начинается с нуля). Тип поля определяет имя метода</td></tr>
<tr><td>char GetChar(intordinal)</td></tr>
<tr><td>bool GetBoolean(intordinal)</td></tr>
<tr><td>System.DateTime GetDateTime(intordinal)</td></tr>
<tr><td>bool NextResult()</td><td>Переводит курсор перед первой строкой следующего набора результатов, если было прочитано несколько наборов составной SQL-командой SELECT</td></tr>
<tr><td>void Close()</td><td>Закрывает модуль чтения</td></tr>
</table>

Пример 1. Просмотр таблицы с помощью объектов **DbCommand** и **DbDataReader**

Когда данных для чтения много или они требуют длительной обработки, рационально сначала считать все данные, затем закрыть соединение и после этого обрабатывать данные в отключенном режиме.

Создайте командой File/New/Project новое решение с именем ADO и проект WinForms1 для первого упражнения

Поместите на форму из панели Toolbox нужные компоненты и настройте их в соответствии с таблицей свойств

Компонент	**Свойство**	**Значение**
Form	Text	Упражнение 1
DataGridView	(Name)	myDataGrid
	Dock	Top
	AutoSizeColumnsMode	Fill
Button	(Name)	btnPopulate
	Text	Заполнить
	AutoSize	true
Button	(Name)	btnDataBind
	Text	Связать
	AutoSize	true
Label	(Name)	lblRegCount
	Text	Количество записей таблицы:

В панели Solution Explorer создайте папку Data для корневого узла проекта командой **Add/New Folder**, вызовите для нее контекстное меню и скопируйте из прилагаемой папкиSource файлNorthwind.mdbкомандой **Add/Existing Item**. При появлении мастера Data Source Configuration Wizard отмените его кнопкой **Cancel.**

Выделите в Solution Explorer файл Northwind.mdb и в панели **Properties** проверьте, что его свойства имеют значение

Build Action=Content

Copy to Output Directory=Copy always

Создайте обработчик для кнопки btnPopulate и btnDataBind

Внесите изменения в файл Form1.cs, выделенные в листинге

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
```

```
using System.Text;
using System.Windows.Forms;
// Дополнительные пространства имен
using System.Data.OleDb;
using System.Data.Common;
using System.Collections;
namespace WinForms1
{
public partial class Form1 : Form
{
public Form1()
{
InitializeComponent();
// Отключаем кнопку 'Связать'
btnDataBind.Enabled = false;
// Отключить системную кнопку
this.MaximizeBox = false;
}
// Создаем объект массива списков
ArrayList dbRecordsList = new ArrayList();// Поле
// Строка соединения с абсолютным путем к БД, определяемым
сборкой
String ConnectionString(String fileName)
{
string JetEngineString = @"Provider=Microsoft.Jet.OLEDB.4.0;"
+ "Jet OLEDB:Engine Type=5;Data Source=";
string pathToFile = Application.StartupPath.ToString() + "\\Data\\";
return JetEngineString + pathToFile + fileName.Trim() + ".mdb";
}
// Заполнение массива списков данными
private void btnPopulate_Click(object sender, EventArgs e)
{
// Создаем объект соединения
OleDbConnection objConnection =
new OleDbConnection(ConnectionString("Northwind"));
dbRecordsList.Clear();// Очищаемсписок (необязательно!)
// После выполнения тела инструкции using освободится подключение
using (objConnection)
{
// Создаем объект команды с последующими настройками
OleDbCommand objCommand = new OleDbCommand();
objCommand.CommandType = CommandType.Text;// Необязательно,
поумолчанию
objCommand.CommandText = "SELECT * FROM Customers";
objCommand.Connection = objConnection;
```

```
// Или сразу одним конструктором
//OleDbCommand objCommand = new OleDbCommand("SELECT *
FROM Customers", objConnection);
objConnection.Open();// Открываем соединение
// Выполняем объект DbCommand одним из его методов
OleDbDataReaderdataReader =
objCommand.ExecuteReader( // Заполнитьнаборданных
CommandBehavior.CloseConnection);// Отработать и сразу закрыть
соединение
// Счетчик количества записей
intrecCount = 0;
if (dataReader.HasRows)// Флагподнят, еслиестьстроки
{
// Перебрать все строки набора и добавить в коллекцию списка
foreach (DbDataRecord rec in dataReader)
{
dbRecordsList.Add(rec);
recCount++;// Не будем использовать, подсчитаем ниже командой
}
}
// Подсчитываем число записей в таблице
objCommand = new OleDbCommand("SELECT COUNT(*) FROM
Customers", objConnection);
objConnection.Open();
recCount = (int)objCommand.ExecuteScalar();
objConnection.Close();
lblRegCount.Text += recCount.ToString();
}
// Переключаемдоступностькнопок
btnPopulate.Enabled = false;
btnDataBind.Enabled = true;
}
private void btnDataBind_Click(object sender, EventArgs e)
{
// Связываем список с элементом отображения
myDataGrid.DataSource = dbRecordsList;
btnDataBind.Enabled = false;// Отключаем кнопку
}
}
}
```

Запустите приложение, чтобы получить результат

DbDataReader, возвращаемый методом objCommand.ExecuteReader(), позволяет читать данные в соответствии с исполняемой командой SELECT построчно в однонаправленном, доступном только для чтения потоке. Применение DbDataReader - простейший *путь* извлечения данных из

хранилища, но ему недостает возможностей сортировки и связывания автономного DataSet.

DbDataReader не позволяет выполнять *обратный* просмотр набора результатов и редактировать их. Это лишь способ последовательного чтения в одном направлении, использующий постоянное подключение к *БД*.

В соответствии со значением перечисления CommandBehavior. CloseConnection соединение с *БД* закрывается сразу же, как только *объект* команд закончит чтение данных в DbDataReader. Но для подстраховки в нашем приложении *модуль* чтения и подключение закрываются автоматически после завершения инструкции using.

В отключенном режиме загруженные данные находятся в объекте ArrayList и показываются пользователю после связывания cDataGridView. *Объект* DataGridView, в нашем случае, показывает данные в режиме только для чтения, поскольку 'знает', чтоArrayList не является редактируемым.

Задания

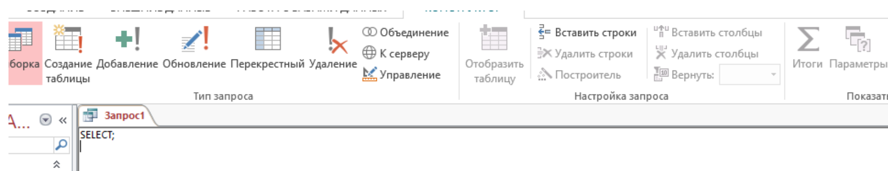
1 Изучить теоретические сведения.

2 В соответствии с вариантом задания организовать выборки данных с помощью запросов:

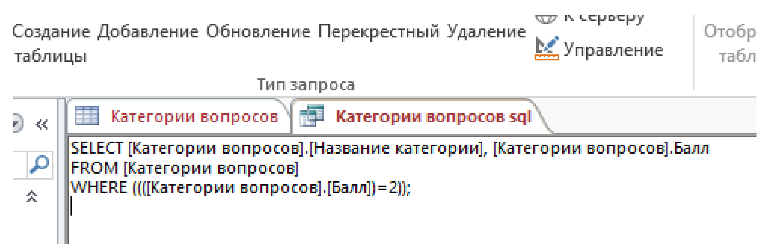
- по одной таблице;
- нескольким таблицам;
- с использованием логических операций;
- с использованием агрегатных функций.

Порядок выполнения работы

Для создания запроса в режиме SQL, выбрать Создание, Конструктор запросов, затем отменить добавление таблицы, перейти в режим SQL.

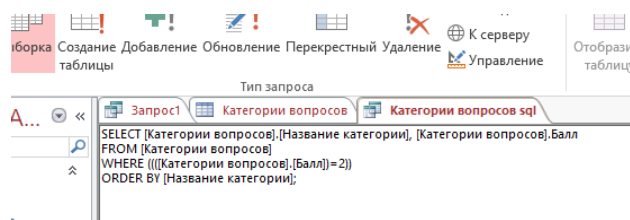


В строке SELEKT перечислить поля, которые необходимы для запроса, причем если в имени поля имеется пробел, то его берем в квадратные скобки, далее следует зарезервированное слово FROM, имя таблицы, условие отбора WHERE и условие.

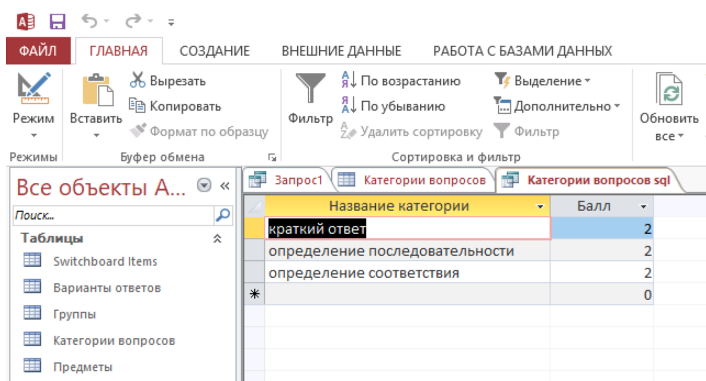


В запросе по нескольким таблицам в строке SELEKT перечислить поля нужных таблиц, которые необходимы для запроса, причем если имя поля одинаковое в разных таблицах, то используем явное определение полей, если поле имеет уникальное имя, то использовать явное определение полей не обязательно. Далее следует зарезервированное слово FROM, имена таблиц, условие отбора WHERE и условие. Может быть несколько условий, тогда необходимо связать их с помощью логических операций NOT, AND или OR.

Добавим сортировку



И результат:



В запросе с агрегатными функциями воспользуемся таблицей Предметы и найдем количество предметов каждого преподавателя, для этого напишем код:

```
SELECT [Код преподавателя], COUNT([Код предмета]) AS [Количество предметов]
FROM Предметы
GROUP BY [Код преподавателя];
```

Здесь Количество предметов – это заголовок поля, в которое помещается результат вычислений. Если не вставлять в запрос группировку GROUP BY [Код преподавателя], то будет посчитано общее количество

предметов, а в результате такого запроса вычисления производятся для каждого преподавателя.

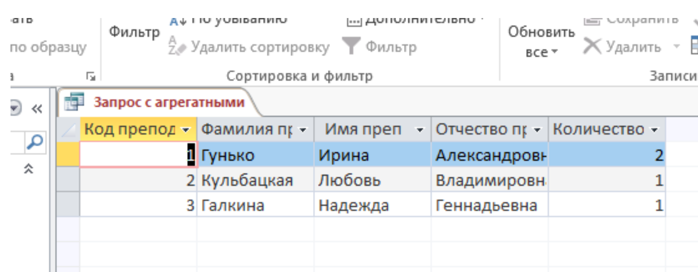
Добавим к запросу ФИО преподавателей из таблицы Преподаватели, для чего в список полей добавим поля из указанной таблицы, а в строку FROM саму таблицу:

SELECT Предметы.[Код преподавателя], COUNT([Код предмета]) AS [Количество предметов]
FROM Предметы, Преподаватели
GROUP BY [Код преподавателя];

Возникает проблема, для каждого кода преподавателя в запросе перечисляются все записи из таблицы Преподаватели. Чтобы устранить этот недочет, необходимо добавить условие отбора Предметы.[Код преподавателя]=Преподаватели.[Код преподавателя] с помощью условия, которое записывается с помощью предложения HAVING Предметы.[Код преподавателя]=Преподаватели.[Код преподавателя]. А для того, чтобы запрос выполнил предложенные действия, необходимо включить поля, не участвующие в функции COUNT, в список полей GROUP BY:

SELECT Предметы.[Код преподавателя], [Фамилия преп], [Имя преп], [Отчество преп], COUNT([Код предмета]) AS [Количество предметов]
FROM Предметы, Преподаватели
GROUP BY Предметы.[Код преподавателя], Преподаватели.[Код преподавателя], [Фамилия преп], [Имя преп], [Отчество преп]
HAVING Предметы.[Код преподавателя]=Преподаватели.[Код преподавателя];

Получим желаемый результат:



Код препод	Фамилия пр	Имя преп	Отчество пр	Количество
1	Гунько	Ирина	Александровн	2
2	Кульбацкая	Любовь	Владимировн	1
3	Галкина	Надежда	Геннадьевна	1

Содержание отчета

1 Название работы
2 Цель работы
3 Перечень технических средств обучения
4 Порядок выполнения работы
5 Ответы на контрольные вопросы
6 Вывод

Варианты заданий

Варианты заданий представлены в практической работе № 39
Контрольные вопросы:

1. Что такое SQL?
2. Возможности SQL запросов.
3. Состав любого оператора SQL.
4. Соглашения SQL.
5. Состав запроса в режиме SQL.
6. Для чего используются приложения в Ms Access?
7. Инструменты создания приложений в Ms Access.
8. Какие современные технологии используются при создании приложений в Ms Access?
9. Принципы технологий, применяемых в Ms Access.

Используемая литература

– Г.Н.Федорова Основы проектирования баз данных. М.: Академия, 2020
– Г.Н.Федорова Разработка, администрирование и защита баз данных. М.: Академия, 2018
– https://codetown.ru/category/sql